COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MASTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIONES

MASTER'S FINAL PROJECT

Development and Simulation of a Vehicular Communication System for Emergency Vehicle Detection

Author: Nicolás Corsini Santolaria

Supervisor: Carol Davids

Co-Supervisor: Alvin Chin

Chicago

I declare, under my responsibility, that the Project presented with the title

Development and Simulation of a Vehicular Communication System for Emergency Vehicle Detection at the School of Engineering – ICAI of the Universidad Pontificia Comillas in the academic year 2024/25 is my own work, original and unpublished, and has not been previously submitted for any other purpose.

The Project is not a plagiarism of another, either in whole or in part, and the information taken from other documents is duly referenced.

Signed.: Nicolás Corsini   Date: 18/ 08/ 2025

[illegible]

Authorized for submission of the project

Prof. Carol Davids

Signed.: Carol Davids   Date: 30/ 08/ 2025

Carol Davids

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Desarrollo y simulación de un sistema de comunicación vehicular para la detección de vehículos de emergencia

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2024/25 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Nicolás Corsini Fecha: 18/ 08/ 2025

[illegible]

Autorizada la entrega del proyecto

Prof. Carol Davids

Fdo.: Carol Davids Fecha: 30/ 08/ 2025

Carol Davids

MASTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIONES

MASTER'S FINAL PROJECT

Development and Simulation of a Vehicular Communication System for Emergency Vehicle Detection

Author: Nicolás Corsini Santolaria

Supervisor: Carol Davids

Co-Supervisor: Alvin Chin

Chicago

DESARROLLO Y SIMULACIÓN DE UN SISTEMA DE COMUNICACIÓN VEHICULAR PARA LA DETECCIÓN DE VEHÍCULOS DE EMERGENCIA

Autor: Corsini Santolaria, Nicolas
Director: Davids, Carol
Co-director: Chin, Alvin
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Este proyecto presenta el diseño y la simulación de un sistema de comunicación vehicular capaz de alertar a los conductores sobre la presencia de una ambulancia que se aproxima en tiempo real. Mediante el uso de comunicaciones dedicadas de corto alcance (DSRC) y de los mensajes de seguridad básicos (BSM), el sistema supera las limitaciones de las sirenas y luces tradicionales, garantizando una detección más temprana y fiable. Los resultados de simulación demuestran su eficacia en condiciones urbanas realistas, incluso en presencia de propagación multitrayecto, efectos Doppler y ruido.

Palabras clave: Telecomunicaciones vehiculares, DSRC, V2V, Pérdidas multitrayecto, seguridad vial.

1. Introducción

Las ciudades modernas se enfrentan a un reto creciente en la gestión del tráfico y la movilidad urbana, especialmente en lo que respecta al paso de los vehículos de emergencia. Cada segundo cuenta cuando una ambulancia debe atravesar calles congestionadas para llegar a su destino, y la capacidad de advertir a los conductores con suficiente antelación puede salvar vidas. Este proyecto aborda este problema mediante el diseño de un sistema de comunicaciones vehiculares que permite a los automóviles recibir en tiempo real información precisa sobre la ubicación y dirección de las ambulancias cercanas, con el fin de mejorar la seguridad vial y facilitar el trabajo de los servicios de emergencia.

2. Definición del proyecto

Este proyecto desarrolla y simula un sistema de comunicación vehicular diseñado para alertar a los conductores sobre la aproximación de ambulancias en tiempo real. Utilizando MATLAB, se ha implementado una simulación completa de la capa física, modelando la transmisión, la recepción y la ecualización en condiciones urbanas realistas que incluyen multitrayectoria, efectos Doppler y ruido. El objetivo principal es demostrar que la detección fiable de ambulancias puede lograrse utilizando comunicaciones DSRC en la banda de 5.9 GHz, garantizando que los vehículos reciban alertas oportunas sobre la presencia de servicios de emergencia cercanos. Los objetivos adicionales incluyen evaluar el rendimiento del sistema en diferentes escenarios, comprobar la robustez del algoritmo de ecualización y diseñar una aplicación orientada al usuario capaz de mostrar la posición relativa y la trayectoria de la ambulancia de una manera clara y práctica para los conductores.

3. Descripción del modelo/sistema/herramienta

El sistema diseñado se basa en una arquitectura modular que incluye bloques para la generación de mensajes de seguridad básicos (BSM), su codificación y modulación, la transmisión a través de un canal urbano con multitrayectoria y ruido, y la posterior recepción, ecualización y decodificación. Se han implementado técnicas de modulación OFDM con BPSK, junto con un algoritmo de estimación de canal iCDP, lo que permite compensar eficazmente las distorsiones introducidas por la propagación multitrayectoria, el ruido y los efectos Doppler. El diseño también contempla un proceso de filtrado que permite discriminar coordenadas no válidas cuando un BSM se recibe con errores, garantizando que el sistema pueda reconstruir trayectorias coherentes incluso cuando solo una fracción de los mensajes se recibe correctamente.

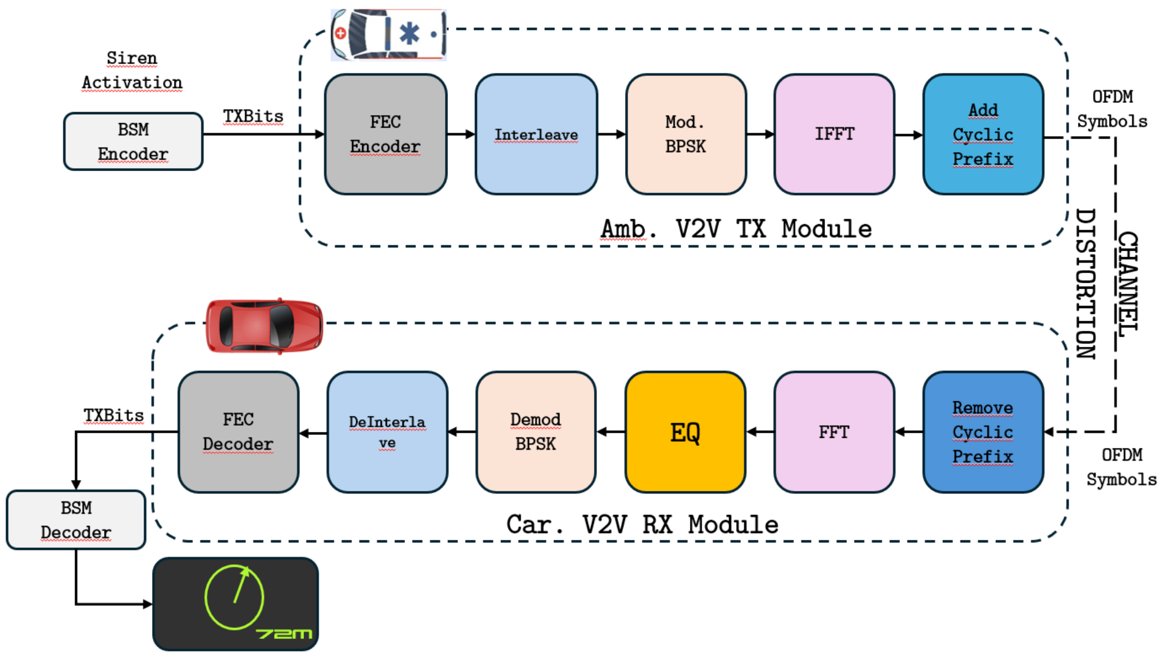
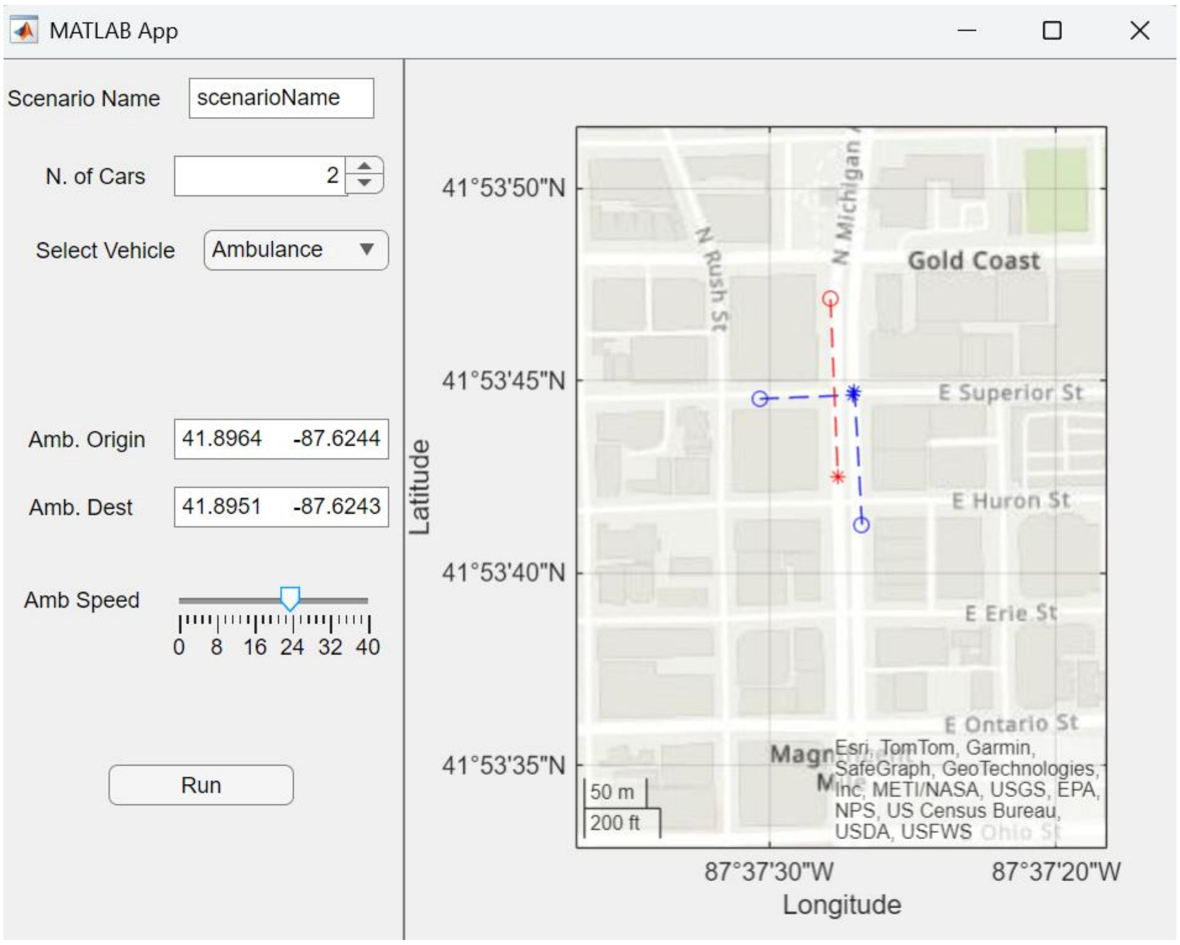


Image 1. Diagrama de bloques

Al finalizar el proyecto, se han creado dos aplicaciones en MATLAB utilizando App Designer.

La primera permite generar y simular escenarios personalizados, en los que se pueden colocar ambulancias y vehículos en un entorno urbano simulado, especificando velocidades y trayectorias.


MATLAB App

Scenario Name: scenarioName

N. of Cars: 2

Select Vehicle: Ambulance

Amb. Origin: 41.8964 -87.6244

Amb. Dest: 41.8951 -87.6243

Amb Speed

Image 2. App de creación de scenarios

La segunda aplicación se centra en la visualización de resultados, ofreciendo a los usuarios una interfaz interactiva que muestra la posición estimada de la ambulancia en tiempo real desde la perspectiva de cada vehículo, así como métricas de rendimiento como la tasa de error de bits.

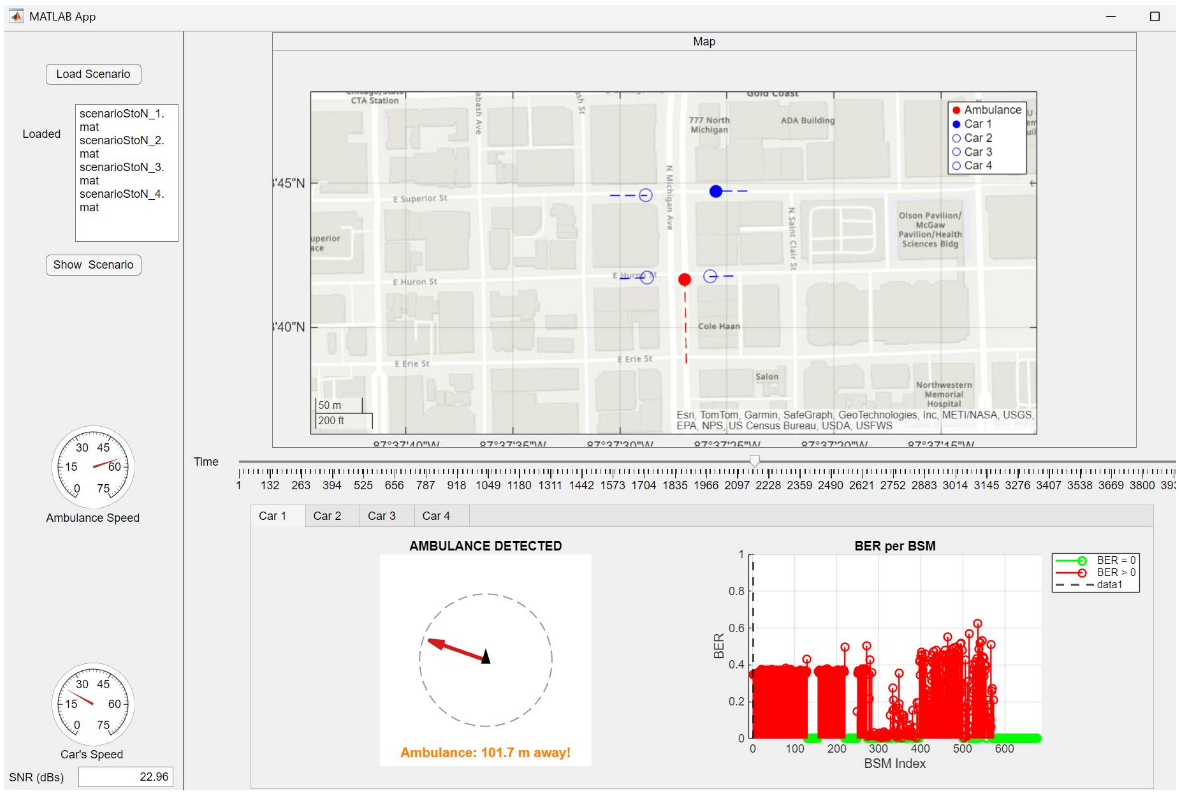


Image 3. App de visualización de resultados

4. Resultados

Las simulaciones extensivas realizadas con el sistema diseñado han demostrado su eficacia en una amplia variedad de escenarios urbanos. El sistema es capaz de alertar de manera fiable a los conductores sobre la presencia de una ambulancia que se aproxima hasta una distancia máxima de 200 metros, siempre que exista una relación señal-ruido mínima de 10 dB. Dentro de un radio de 100 metros, el 100% de los vehículos evaluados fueron alertados con éxito en todos los escenarios, mientras que este porcentaje se mantuvo en un 90% para un radio de 150 metros. El sistema también demostró ser robusto frente a la propagación multitrayectoria, los efectos Doppler y el ruido, manteniendo un rendimiento fiable con velocidades de hasta 90 mph. Estos resultados confirman que el diseño propuesto cumple con los objetivos del proyecto y demuestran la viabilidad de los sistemas de comunicación vehicular para la detección de vehículos de emergencia en condiciones realistas.

5. Conclusiones

El proyecto demuestra que es técnicamente viable diseñar un sistema de comunicaciones vehiculares capaz de detectar de forma fiable a los vehículos de emergencia en entornos urbanos. A través de simulaciones y aplicaciones interactivas, el sistema ha mostrado su capacidad para alertar a los conductores con gran precisión, robustez frente a las distorsiones del canal y un rendimiento práctico en condiciones realistas. Estos resultados confirman la viabilidad del enfoque propuesto y sientan las bases para futuras implementaciones en el mundo real.

6. Referencias

1. talks.com/technology/dsrc-vs-c-v2x/

2. Cheng, Z. a. (2014). *IEEE 802.11p for V2V Communications.*

3. Guillermo Francia, D. S. (2023). *Basic Safety Messages (BSM) Test Data Generation for Vehicle Security Machine Learning Systems.*

4. Kenney, J. B. (2020). *Dedicated Short Range Communications (DSRC) Standards in the United States.*

5. Mathworks. (n.d.). *comm.RayTracingChannel*. Retrieved from https://www.mathworks.com/help/comm/ref/comm.raytracingchannel-system-object.html

6. Mathworks. (n.d.). *Intersection Movement Assist Using Vehicle to Vehicle Communication*. Retrieved from https://www.mathworks.com/help/driving/ug/intersection-movement-assist-using-v2v.html

7. Mathworks. (s.f.). *OFDM Transmitter and Receiver with BPSK baseband, RF up-down conversion*. Obtenido de https://www.mathworks.com/matlabcentral/fileexchange/57494-ofdm-transmitter-and-receiver-with-bpsk-baseband-rf-up-down-conversion

8. Mathworks. (n.d.). *Raytracing*.

9. Mathworks. (n.d.). *Traffic Light Negotiation Using Vehicle-to-Everything Communication*. Retrieved from https://www.mathworks.com/help/driving/ug/traffic-light-negotiation-using-v2x.html

10. SAE International. (2020). *SAE International J2735 Surfave Vehicle Standard.*

11. Tong Wang, A. H. (2018). *An improved channel estimation technique for IEEE 802.11p Standard in Vehicular Communications.*

DEVELOPMENT AND SIMULATION OF A VEHICULAR COMMUNICATION SYSTEM FOR EMERGENCY VEHICLE DETECTION

Author: Corsini Santolaria, Nicolas
Supervisor: Davids, Carol
Co-supervisor: Chin, Alvin
Collaborating Entity: ICAI-Universidad Pontificia de Comillas

ABSTRACT

This project presents the design and simulation of a vehicular communication system capable of alerting drivers to the presence of an approaching ambulance in real time. By leveraging Dedicated Short-Range Communications (DSRC) and Basic Safety Messages (BSMs), the system overcomes the limitations of traditional sirens and lights, ensuring earlier and more reliable detection. Simulation results demonstrate its effectiveness under realistic urban conditions, even in the presence of multipath propagation, Doppler shifts, and noise.

Keywords: Vehicular Communications, V2V, DSRC, Multipath raytracing, Traffic Safety

1. Introduction

Emergency vehicles such as ambulances, fire trucks, and police cars rely on sirens and flashing lights to warn other drivers, but these traditional methods are often insufficient in dense urban environments. Traffic congestion, soundproof vehicle cabins, and limited lines of sight can delay drivers' reactions, reducing the efficiency of emergency response. To address this challenge, modern vehicular communication technologies provide a new opportunity to deliver reliable, real-time alerts directly to drivers, improving both safety and response times.

2. Project definition

This project develops and simulates a vehicular communication system designed to alert drivers of approaching ambulances in real time. Using MATLAB, a complete physical-layer simulation has been implemented, modeling transmission, reception, and equalization under realistic urban conditions including multipath, Doppler effects, and noise. The main objective is to demonstrate that reliable ambulance detection can be achieved using Dedicated Short-Range Communications (DSRC) in the 5.9 GHz band, ensuring that vehicles receive timely alerts of emergency services nearby. Additional goals include evaluating system performance across different scenarios, testing the robustness of the equalization algorithm, and designing a user-oriented application capable of displaying the ambulance's relative position and trajectory to drivers in a clear and practical way.

3. System and applications designed

The system designed in this project follows a block-based communication structure, where each Basic Safety Message (BSM) is processed through a series of transmission and reception stages. At the transmitter side, the bits are first encoded and modulated using Orthogonal Frequency Division Multiplexing (OFDM) with Binary Phase Shift Keying (BPSK) modulation. These modulated symbols are then transmitted over a channel modeled to include multipath propagation, Doppler effects, and additive white Gaussian noise, reflecting the real conditions of urban vehicular environments. At the receiver, the signal is equalized using the improved Constructed Data Pilots (iCDP) algorithm, which compensates for the effects of fading and time-varying channel distortion, ensuring reliable message recovery even under severe interference. The design ensures that complete message recovery is not strictly necessary, since even a few correctly decoded BSMs are enough to reconstruct the ambulance's trajectory and determine its direction of approach. The block diagram is pictured below:

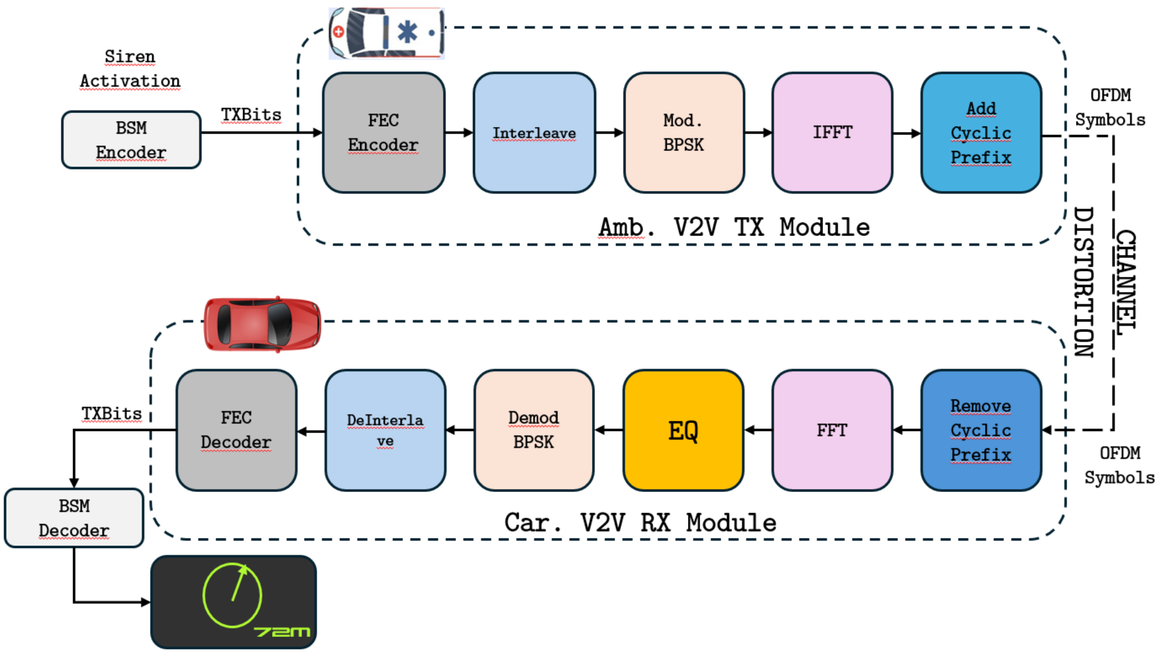


Image 4. Sytem block diagram

To make the system practical and user-friendly, two MATLAB applications were developed.

The first app allows the creation of customizable urban scenarios, where ambulances and cars can be placed with chosen origins, destinations, and speeds, and where channel conditions can be simulated. Its interface is shown below:

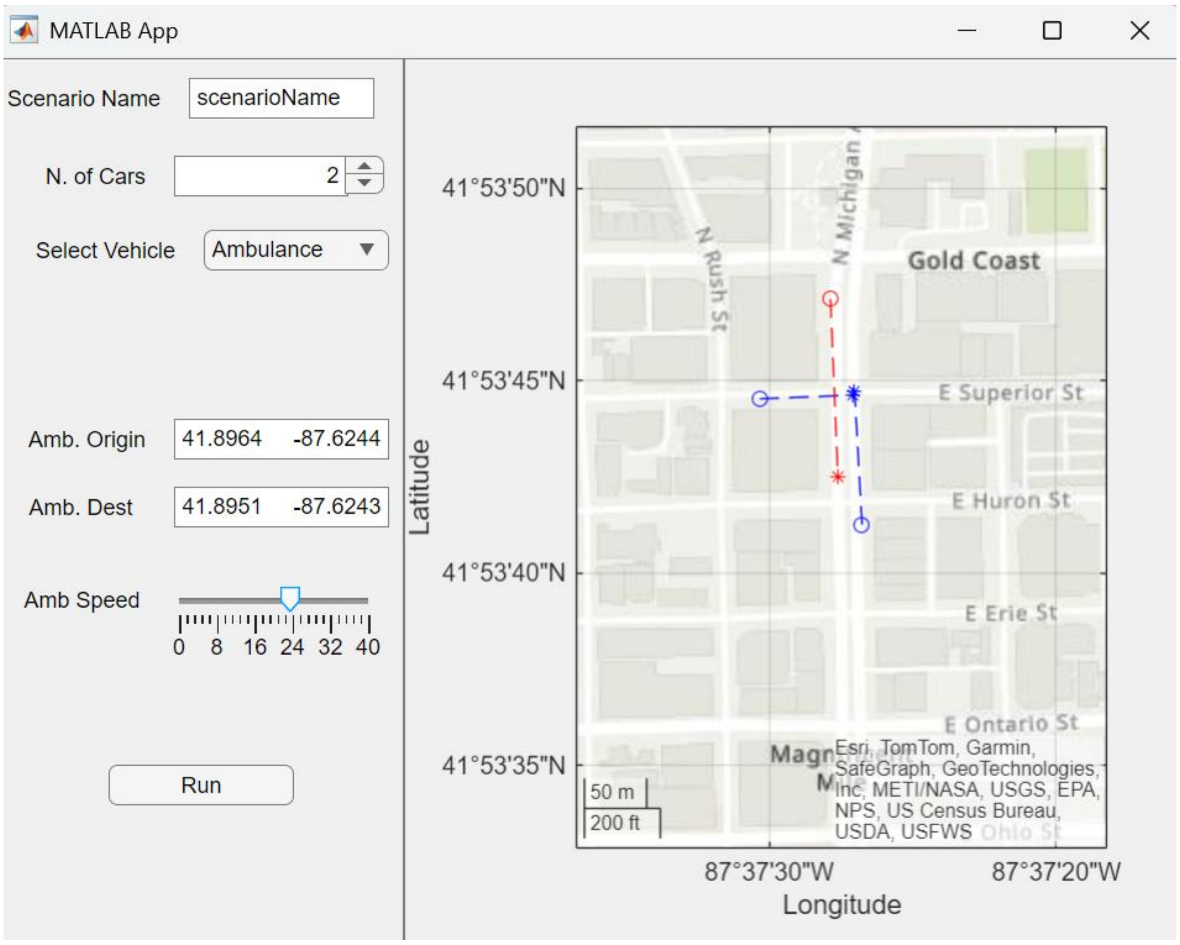


Image 5. Scenario creation App

The second app provides an interactive visualization of the results, showing how drivers would receive real-time alerts on their dashboards, including the relative direction of the ambulance and its distance at any given instant. Together, these applications demonstrate not only the technical robustness of the system but also its potential for real-world integration into next-generation vehicular safety technologies. Its interface is shown below:

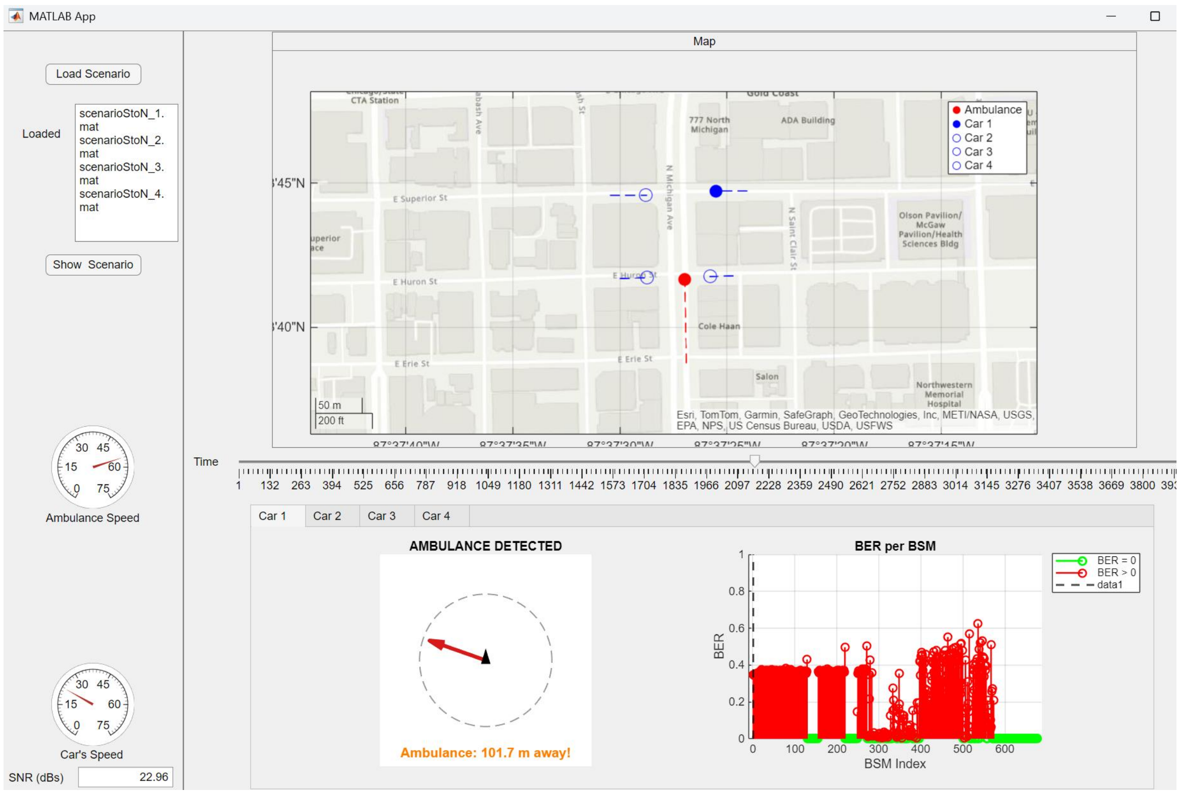


Image 6. Results viewer app

4. Results

Extensive simulations carried out with the designed system have demonstrated its effectiveness under a wide range of urban scenarios. The system is capable of reliably alerting drivers to the presence of an approaching ambulance up to a maximum distance of 200 meters, provided a minimum signal-to-noise ratio of 10 dB. Within a 100-meter radius, 100% of tested vehicles were successfully alerted in all scenarios, while this percentage remained at 90% for a radius of 150 meters. The system also proved robust against multipath propagation, Doppler effects, and noise, maintaining reliable performance for vehicle speeds of up to 90 mph. These results confirm that the proposed design fulfills the project’s objectives and demonstrates the feasibility of vehicular communication systems for emergency vehicle detection in realistic conditions.

5. Conclusions

The project demonstrates that it is technically feasible to design a vehicular communication system capable of reliably detecting emergency vehicles in urban environments. Through simulations and interactive applications, the system has shown its ability to alert drivers with high accuracy, robustness to channel impairments, and practical performance within realistic conditions. These results confirm the viability of the proposed approach and lay the groundwork for future real-world implementations.

6. Referencias

1. talks.com/technology/dsrc-vs-c-v2x/

2. Cheng, Z. a. (2014). *IEEE 802.11p for V2V Communications.*

3. Guillermo Francia, D. S. (2023). *Basic Safety Messages (BSM) Test Data Generation for Vehicle Security Machine Learning Systems.*

4. Kenney, J. B. (2020). *Dedicated Short Range Communications (DSRC) Standards in the United States.*

5. Mathworks. (n.d.). *comm.RayTracingChannel*. Retrieved from https://www.mathworks.com/help/comm/ref/comm.raytracingchannel-system-object.html

6. Mathworks. (n.d.). *Intersection Movement Assist Using Vehicle to Vehicle Communication*. Retrieved from https://www.mathworks.com/help/driving/ug/intersection-movement-assist-using-v2v.html

7. Mathworks. (s.f.). *OFDM Transmitter and Receiver with BPSK baseband, RF up-down conversion*. Obtenido de https://www.mathworks.com/matlabcentral/fileexchange/57494-ofdm-transmitter-and-receiver-with-bpsk-baseband-rf-up-down-conversion

8. Mathworks. (n.d.). *Raytracing*.

9. Mathworks. (n.d.). *Traffic Light Negotiation Using Vehicle-to-Everything Communication*. Retrieved from https://www.mathworks.com/help/driving/ug/traffic-light-negotiation-using-v2x.html

10. SAE International. (2020). *SAE International J2735 Surfave Vehicle Standard.*

11. Tong Wang, A. H. (2018). *An improved channel estimation technique for IEEE 802.11p Standard in Vehicular Communications.*

Index

Figures

Chapter 1. INTRODUCTION

Currently, in the United States, around 6,500 traffic accidents involving an ambulance in emergency service are reported each year. These accidents are particularly severe due to the high speed at which emergency vehicles travel. In fact, 35% of them result in injuries or fatalities, in addition to any person who might be receiving medical attention inside the ambulance itself. Therefore, it is reasonable to ask: what is the cause of such specific accidents, and how could they be reduced?

For many decades, the protocol has been that when an ambulance is in emergency service, it activates its sirens. These sirens have a visual component, with bright, flashing lights, and an auditory component, with the typical siren sound controllable from the cabin. The purpose of the sirens is to alert nearby drivers (and pedestrians) to the presence of the ambulance, so that they can react in time. It is clear that the visual component fulfills its role effectively when necessary, particularly during nighttime scenarios. However, the siren's sound, designed for daytime situations, seems not to be entirely effective.

Indeed, among the driver community, it is very common to hear that it is genuinely difficult to locate an ambulance in broad daylight in an urban environment using only the sirens. This is due to the very nature of sound waves, which bounce off building facades. Additionally, drivers may have their windows up, or the music volume turned up high. In some cases, sirens can even be counterproductive when trying to identify the exact location of an emergency vehicle. It becomes evident, therefore, that sirens are completely ineffective at performing such a critical task.

Ultimately, when an ambulance is in emergency service, every extra second of travel can mean the difference between life and death for a person. It is not just a matter of reducing ambulance accidents, but also of optimizing their urgent journeys through better cooperation from other drivers. However, with such an outdated technology as sirens, this goal remains unattainable.

So, why is such a rudimentary solution still being used for something so crucial? There seems to be no clear answer to this question, but in a society as advanced as ours, it is something that should certainly be reinvented. Therefore, the idea behind this project is to explore a technological solution adapted to today's reality, leveraging telecommunications to provide an effective remedy to a truly relevant problem.

Chapter 2. STATE OF THE ART

In recent years, the automotive and telecommunications industries have worked intensively on the development of new technologies enabling direct communication between vehicles, known as Vehicle-to-Vehicle (V2V) communication. The main goal of this new communication layer is to enhance road safety, reduce accidents, and optimize traffic flow. In the context of this project, it is particularly relevant to understand the current state of V2V technologies, as they form the foundation upon which an effective solution to the problem of ambulances in emergency service can be built.

The first major technological approach in V2V was DSRC (Dedicated Short-Range Communications). This technology, developed primarily in the United States, is based on the IEEE 802.11p standard, an adaptation of traditional Wi-Fi for vehicular environments. DSRC allows vehicles to exchange messages directly with each other, without the need for network infrastructure, using frequencies around 5.9 GHz. Its main advantages lie in its low latency, around 10 milliseconds, and its simplicity of implementation, given that it inherits much of the already widely known and tested Wi-Fi technology. Thanks to its rapid reaction time, DSRC is capable of supporting critical safety applications, such as collision warnings or emergency braking alerts.

However, despite its theoretical advantages, DSRC has encountered several obstacles on its path toward widespread adoption. Its channel access method, based on CSMA/CA (listen before talk), can suffer from message collisions in dense urban environments—precisely where the highest reliability is needed. Furthermore, investments in dedicated DSRC infrastructure have been limited, and the emergence of new technologies has overshadowed its initial momentum. Although numerous pilot programs and test projects were conducted, especially in the United States, the truth is that DSRC did not fully establish itself as the definitive solution for V2V communications.

Given DSRC's limitations, a new proposal emerged: C-V2X (Cellular Vehicle-to-Everything), standardized by the international body 3GPP. C-V2X represents a paradigm shift, as it is not based on Wi-Fi but rather on cellular technology such as LTE and, more recently, 5G. This approach allows vehicles to communicate directly with each other (PC5 mode) or to rely on the network infrastructure to access real-time traffic information (Uu mode). C-V2X promises substantial improvements over DSRC: greater reliability in dense environments, greater communication range (over 1 kilometer in open areas), and better spectrum efficiency, thanks to advanced resource scheduling techniques.

Another important advantage of C-V2X is its natural evolution path towards 5G NR-V2X networks, where even lower latency, higher transmission rates, and new capabilities such as sensor information sharing (e.g., camera and radar data between vehicles) are expected. This growth potential positions C-V2X as the preferred technology for the automotive industry and regulators in Europe, China, and North America. Indeed, some countries like China have decisively embraced C-V2X, even mandating its use in smart city pilot projects.

Therefore, in the current state of V2V technology, there are two coexisting paths: DSRC, which is more mature and simpler, and C-V2X, which represents the future of connected mobility. When designing a modern solution to improve the interaction between ambulances and drivers in an urban environment, it is clear that any development should be based on the principles of direct communication, low latency, and high reliability offered by these V2V technologies. Particularly, the global trend suggests that C-V2X will serve as the foundation for future intelligent transportation systems.

In addition to transmission technologies, another fundamental aspect of V2V communications is the structure and content of the messages exchanged between vehicles. It is not enough to simply transmit signals; it is essential to clearly define what type of information is transmitted, how it is formatted, and for what purpose. In this regard, the use of BSMs (Basic Safety Messages) has been standardized, becoming the core of safety communication within DSRC systems.

A BSM is a periodic message transmitted by each vehicle to inform its surroundings of its position, speed, heading, and other essential dynamic parameters. These messages are designed to be extremely lightweight and quick to process, as their main goal is to enable immediate reaction by nearby vehicles or infrastructure. The format of the BSM is standardized under SAE J2735, ensuring interoperability across different manufacturers and devices.

Structurally, the BSM is divided into two parts:

- Part I contains the essential and mandatory data, such as the vehicle identifier, timestamp, latitude, longitude, elevation, speed, heading angle, and acceleration.

- Part II is optional and may include additional information regarding intended maneuvers, light status, brake status, or other sensor readings.

In practice, for applications such as emergency ambulance detection, Part I of the BSM is more than sufficient. Through the periodic transmission of these messages, a nearby vehicle can precisely and in real time know the location and movement of the ambulance, allowing it to react appropriately even when visual or auditory perception is limited. Moreover, thanks to the small size of the BSM (around 200–300 bits for Part I), its transmission requires minimal bandwidth, making it suitable for dense urban environments without causing network saturation.

Thus, within the current state of V2V technology, BSMs represent a key element: they are the means through which safety communication becomes tangible, reliable, and effective. Any modern solution aiming to optimize the interaction between ambulances and drivers, such as the one proposed in this project, must necessarily rely on the transmission and reception of BSMs to guarantee efficient and standardized operation.

That said, it is important to emphasize that there is currently no specific technology deployed to address the problem targeted by this project. In other words, there is no existing technological solution that allows emergency vehicles to share their GPS coordinates with nearby vehicles. Therefore, this project must leverage and adapt existing technologies to address this critical issue.

Chapter 3. DEFINITION OF WORK

3.1 JUSTIFICATION

The justification for this project arises from the need to address a critical gap in current emergency vehicle detection systems. While existing methods rely primarily on sirens and visual signals, these are increasingly ineffective in modern urban environments. This section outlines the reasons behind the development of the proposed system, identifying the problem, the technological and market gap, and the potential impact of its implementation

3.1.1 PROBLEM IDENTIFICATION

The growing urban density and the increasing number of vehicles on the road have amplified the challenges faced by emergency services, particularly ambulances, in reaching their destinations quickly and safely. Traditional warning systems, such as sirens and flashing lights, are no longer sufficient to guarantee early and accurate detection of emergency vehicles by other drivers. Acoustic signals suffer from significant limitations in urban environments due to sound reflection, insulation from vehicle cabins, and competing noise sources. This often results in delayed reactions from surrounding drivers, increasing the risk of accidents and slowing down emergency response times.

3.1.2 TECHNOLOGICAL AND MARKET GAP

Despite the advances in vehicular communication technologies, no commercially deployed system currently allows emergency vehicles to automatically broadcast their real-time position and trajectory to nearby cars in a standardized and interoperable manner. The lack of such a system represents both a safety gap and a market opportunity. Modern Vehicle-to-Vehicle (V2V) communication protocols, particularly those based on C-V2X and DSRC, offer the low latency, high reliability, and direct connectivity required to address this problem. By leveraging these technologies, it is possible to develop a solution capable of transmitting Basic Safety Messages (BSM) containing an ambulance's GPS coordinates to all surrounding vehicles in real time, regardless of environmental noise or visibility conditions.

3.1.3 PROJECT RELEVANCE AND IMPACT

From a technical perspective, the proposed system builds on existing international standards (IEEE 802.11p and SAE J2735), ensuring compatibility with future intelligent transportation infrastructures. Its implementation in a realistic MATLAB simulation environment allows precise evaluation of performance under urban propagation conditions, including multipath effects, Doppler shifts, and signal degradation caused by buildings. From a market perspective, this project addresses an urgent demand in the automotive and public safety sectors. Its adoption would not only reduce accidents involving emergency vehicles but also improve the efficiency of their journeys, directly contributing to saving lives.

3.2 OBJECTIVES

Several mandatory objectives must be established for this project to be considered a success. These are described below:

The first objective is simply to research and study the entire framework of current vehicular telecommunications. By the end of this project, there must be a clear understanding of the existing technologies and how they are used.

The second objective is to develop a simulation program in MATLAB that can simulate communication between an ambulance and multiple nearby vehicles in an urban environment, following the protocols used in real-world applications. Therefore, this program must simulate an urban setting with realistic buildings and all the distortion effects that typically occur in city communications. With this analysis, the project should be able to assess the feasibility of communication links by analyzing performance metrics such as SNR (Signal-to-Noise Ratio) and BER (Bit Error Rate).

The third objective is to understand and implement the standardized vehicular communication protocol that defines the structure of the transmitted messages. These instructions are provided by the 802.11p standard used in DSRC, as well as by the information outlined in SAE J2735. Following these standards, the goal is to successfully generate a BSM (Basic Safety Message) with the correct structure, where each field in the BSM is described by a specific number of bits. Furthermore, this objective also aims to refine the transmission simulation program by adding the necessary headers and complying with the timing requirements established by the Wi-Fi 802.11p standard.

Finally, the fourth objective seeks to integrate a complete solution in the form of MATLAB code, in which an ambulance and a car in motion can be placed within an urban environment. Once the simulation starts, the ambulance will transmit its coordinates using the message structure described in the third objective, through the simulated channel developed in the second objective. The car will receive the bit sequence and reconstruct the ambulance's coordinates. Lastly, the program must generate a small interface that displays what the driver of the car would see—that is, a simple screen indicating the direction and distance to the ambulance.

Figure 1 shows what the expected output should look like, and what the driver of the car should see on the dashboard.

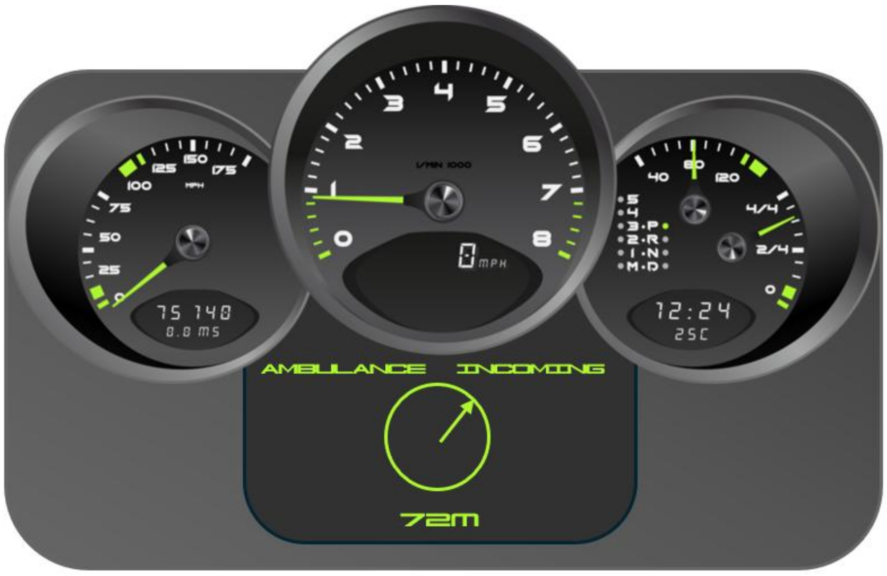


Figure 1. Expected output

The final output should consist of a circular compass that points with an arrow towards the incoming ambulance, with a distance metric just below.

3.3 METHODOLOGY

To achieve the objectives described, the project will be divided into several phases, which are detailed below.

The first phase (Phase 0) consists solely of conducting a study of the state of the art in vehicular communications. It will also involve researching whether any existing solution has already been implemented that allows emergency vehicles to share their GPS coordinates with nearby cars. Completing this phase will fulfill the first objective of the project.

The second phase (Phase 1) involves carrying out an exhaustive analysis of the feasibility of communication links. In other words, during this phase, a program will be developed to simulate communication between an ambulance and several nearby vehicles within a realistic urban environment. This includes simulating distortions caused by effects such as signal reflections, the Doppler effect, and noise. By the end of this phase, a system design should be achieved that enables precise communication (very low BER) between two moving vehicles with a sufficient maximum range (for example, 200 meters). Completing this phase will meet the second objective of the project.

The third phase (Phase 2) focuses on applying to the previous system all the protocols for generating BSM (Basic Safety Messages), in which the ambulance's GPS coordinates are encoded. Therefore, this phase is centered on the generation of the bits to be transmitted. Completing this phase will fulfill the third objective of the project.

Finally, the fourth phase (Phase 3) aims to deliver the final solution with the entire system integrated. This consists of a complete MATLAB code that allows the placement of an ambulance and multiple vehicles within a simulated urban environment, assigning them movement speeds. When the simulation starts, the ambulance transmits its moving GPS coordinates, encoded in BSM format following the SAE J2735 standard. These bits are then modulated into a radio signal according to the IEEE 802.11p standard and transmitted through the simulated urban communication channel in real time. The vehicle receives the modulated signal and processes it in the receiver module, recovering the ambulance's GPS coordinates. Finally, a small screen interface is created, showing an arrow pointing toward the ambulance's location from the car, along with a distance metric indicating how far away it is in that direction. Completing this final phase will achieve the fourth objective and complete the project.

The entire project will be developed exclusively in MATLAB. This tool offers numerous libraries dedicated to telecommunications simulation. Through the use of Toolboxes (library packages), it is possible to access a wide range of resources that enable the completion of the simulations required for this project.

Specifically, the project will primarily utilize the Communications Toolbox, which has already been extensively used during the ICAI coursework. For certain tasks within the context of vehicular telecommunications, the Vehicle Network Toolbox will also be employed, as it is available in MATLAB. To simulate a realistic urban scenario, the SiteViewer function in MATLAB will be used, incorporating a public OpenStreetMap (OSM) file obtained from the internet. To simulate signal propagation, propagation models available within the Communications Toolbox will be applied. In particular, Ray Tracing techniques will be used to study how signals reflect off obstacles in the environment.

Additionally, all information regarding the communication protocols to be followed will be sourced from the official documents of the IEEE 802.11p standard and the official website of SAE J2735. These documents are publicly available and can be accessed openly online.

3.4 PLANIFICATION AND ESTIMATED COST

3.4.1 PLANIFICATION

The execution of the project has been organized into four main phases, each with specific objectives and a clearly defined timeframe, as shown in the planification diagram. The work begins in November 2024 and concludes in August 2025, following a sequential structure with certain overlaps between phases to optimize time and resources. This approach ensures that while one phase is being finalized, preparatory work for the next phase can already begin, thus avoiding unnecessary delays. The project's timeline is displayed in the following Figure 2.

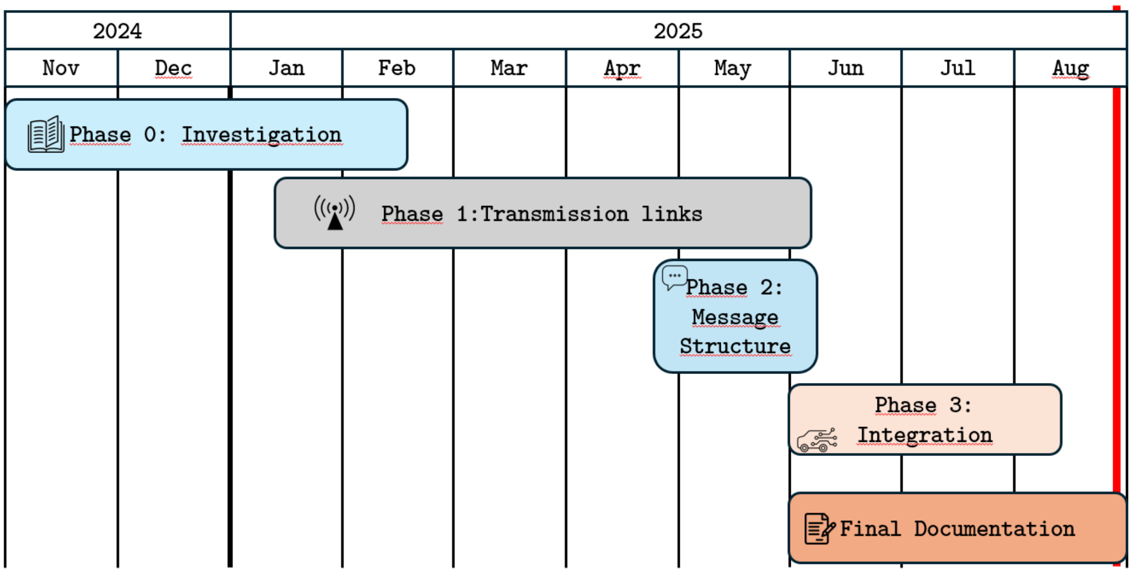


Figure 2. Project Timeline

The first stage, Phase 0: Investigation, extends from November 2024 to February 2025 and focuses entirely on researching the state of the art in vehicular communications. During this period, the existing technologies, standards, and potential solutions are analyzed in depth, with the goal of determining whether any system already addresses the problem targeted by this project. This phase is crucial, as it provides the theoretical and technical foundation for the subsequent development work.

Once the initial research is complete, Phase 1: Transmission Links begins in January 2025 and continues until May 2025. In this phase, the primary objective is to design and simulate the communication channel between an ambulance and surrounding vehicles, within a realistic urban environment. The simulation accounts for common phenomena in city communications, such as multipath propagation, Doppler effect due to vehicle movement, and noise. The aim is to achieve a reliable transmission system capable of operating under the constraints and challenges of real-world conditions.

Following the establishment of a functional communication channel, Phase 2: Message Structure takes place between May and June 2025. This stage involves implementing the standardized Basic Safety Message (BSM) format as defined by SAE J2735, along with the transmission protocols specified by IEEE 802.11p. The correct structuring, encoding, and periodic transmission of these messages is integrated into the communication simulation developed in the previous phase.

The final development stage, Phase 3: Integration, spans from June to August 2025. During this period, all components—channel simulation, message structure, and vehicle positioning—are combined into a single, cohesive system. The ambulance's coordinates are transmitted, processed by the receiving vehicle, and displayed through a simple driver interface showing the distance and direction to the emergency vehicle. This integration ensures that the system can be evaluated as a complete solution within the MATLAB environment.

In parallel with the integration phase, work on the Final Documentation is carried out from June to August 2025. This involves compiling all results, methodologies, and conclusions into the final report, ensuring that the project is thoroughly documented and ready for submission.

3.4.2 ESTIMATED COST

Regarding the estimated cost, since all development is conducted entirely in MATLAB and relies exclusively on simulations, the project does not require the acquisition of physical hardware or additional software beyond what is already available through the university. As a result, the total execution cost is €0, making the project economically viable while still addressing a critical and highly relevant problem in urban mobility and emergency response.

While the cost of the simulation stage of this project is effectively €0, a real-world implementation would require some additional hardware to be installed in vehicles. Modern cars already include many of the necessary components, such as GPS receivers, onboard computing systems, and power supply infrastructure, so these are not included in the cost estimate.

The main additional component required is a V2X radio operating in the 5.9 GHz band, using either DSRC/IEEE 802.11p or C-V2X technology. In large-scale production, these radios add approximately 160 to 170 USD to the manufacturing cost of a vehicle. For aftermarket or small-scale installations, the price can be higher due to packaging, certifications, and distribution.

An external 5.9 GHz antenna is also necessary to ensure proper communication performance. Simple low-profile antennas can cost between 25 and 50 EUR, while more ruggedized vehicular antennas from specialized suppliers may range from 100 to 150 EUR.

Installation costs depend on the local market, but professional vehicle electronics installers typically charge between 100 and 200 USD per hour, and installation would likely take between one and two hours per vehicle.

For a small-scale pilot project, equipping a vehicle with the necessary additional components would therefore cost approximately 300 to 500 EUR per unit, including hardware and installation. In larger deployments, the cost per vehicle could be reduced through bulk purchasing and integrated manufacturing.

Chapter 4. DESCRIPTION OF TECHNOLOGIES

4.1 PROPAGATION IMPAIRMENTS IN URBAN COMMUNICATIONS

Wireless communication between moving vehicles in urban environments is strongly affected by two physical phenomena that make reliable low-latency links challenging: multipath propagation and the Doppler effect. Both produce time- and frequency-varying distortions on the transmitted signal that must be considered when designing and simulating any V2V system.

4.1.1 MULTIPATH PROPAGATION

In urban scenarios radio signals rarely travel only by the direct line-of-sight. Instead, the transmitted wave is split into many components by reflections, diffractions and scattering produced by buildings, vehicles, street furniture and other objects. Each of those components (or "rays") follows a different geometric path to the receiver and therefore arrives with a different delay, amplitude and phase. The superposition of these delayed components at the receiver produces *multipath fading*: depending on the relative delays and phases, the components can add constructively or destructively, producing deep fades at specific frequencies and rapid fluctuations of the received power as geometry changes. In dense urban areas the arrival of many comparable-strength paths yields frequency-selective fading (different frequencies experience different attenuation) and time variations governed by the motion of vehicles and reflectors. These effects are the cause of inter-symbol interference (ISI) in narrowband systems and of frequency-selective channel response in wideband systems, and they are why link performance can vary strongly over short distances and short times. This phenomenon is reflected in Figure 3.

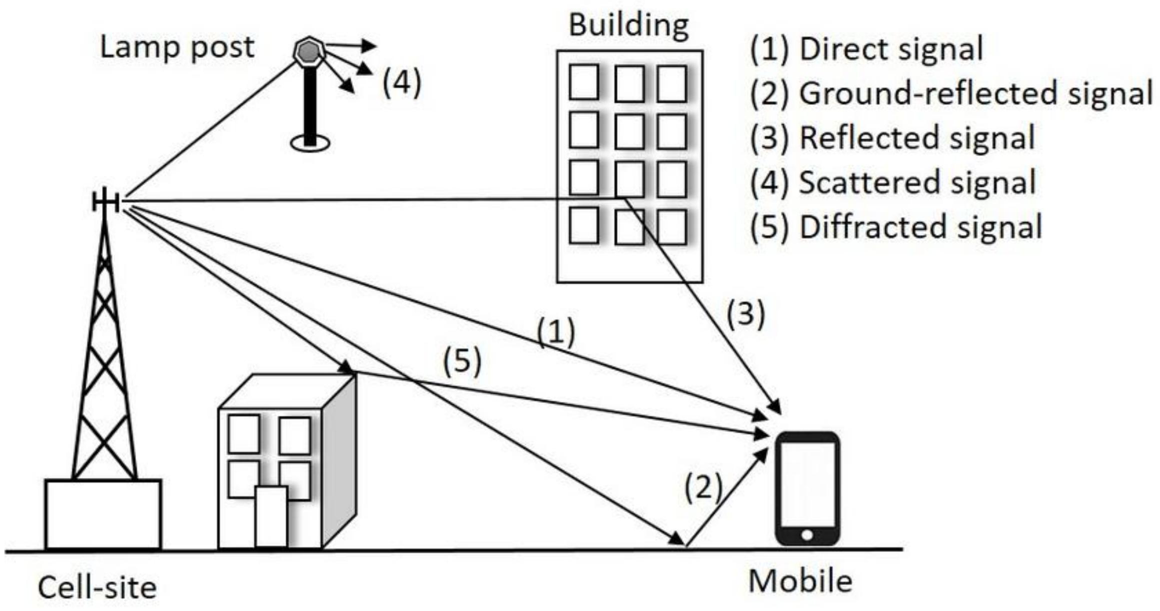


Figure 3 Multipath Propagation in Urban environment (Cheng, 2014)

This figure illustrates the different propagation mechanisms that contribute to multipath in an urban environment. The direct signal (1) travels along the line-of-sight between the transmitter and receiver, while other components are created by interactions with the environment. The ground-reflected signal (2) bounces off the street surface, the reflected signal (3) originates from building facades, and the scattered signal (4) is generated by smaller objects such as lamp posts or street signs, which redirect portions of the wave in multiple directions. Finally, the diffracted signal (5) results from the bending of the wavefront around large obstacles, such as building edges. Each of these components arrives at the receiver with a different delay, amplitude, and phase, producing the constructive and destructive interference patterns that define multipath fading in dense urban scenarios.

From a simulation and design point of view, multipath is commonly modelled either with statistical channel models (e.g., tapped delay lines, power delay profiles) or with geometry-based ray-tracing that computes individual ray delays and angles. In this project, the second option will be employed.

4.1.2 DOPPLER EFFECT

When either transmitter, receiver or reflectors are moving, each arriving path suffers a frequency shift proportional to the relative radial velocity between the wavefront and the receiver. This is the electromagnetic analogue of the familiar Doppler effect for sound: an approaching source creates a higher observed frequency, while a receding one creates a lower frequency. For a carrier frequency f_c and a relative speed component v_{radial} (positive when moving toward the receiver), the Doppler shift for a single path is approximately:

$$f_d = \frac{v_{radial}}{c} f_c$$

where c is the speed of light. In a multipath environment each path generally has a different angle and hence a different radial velocity component, so the channel exhibits a *Doppler spread* (a distribution of shifts across arriving paths). The Doppler spread determines the channel's *coherence time* — how fast the channel impulse response changes — and therefore constrains how quickly the system must update estimates (for synchronization, channel estimation and equalization) to maintain reliable communications. This Doppler effect is better visualized in the following Figure 4.

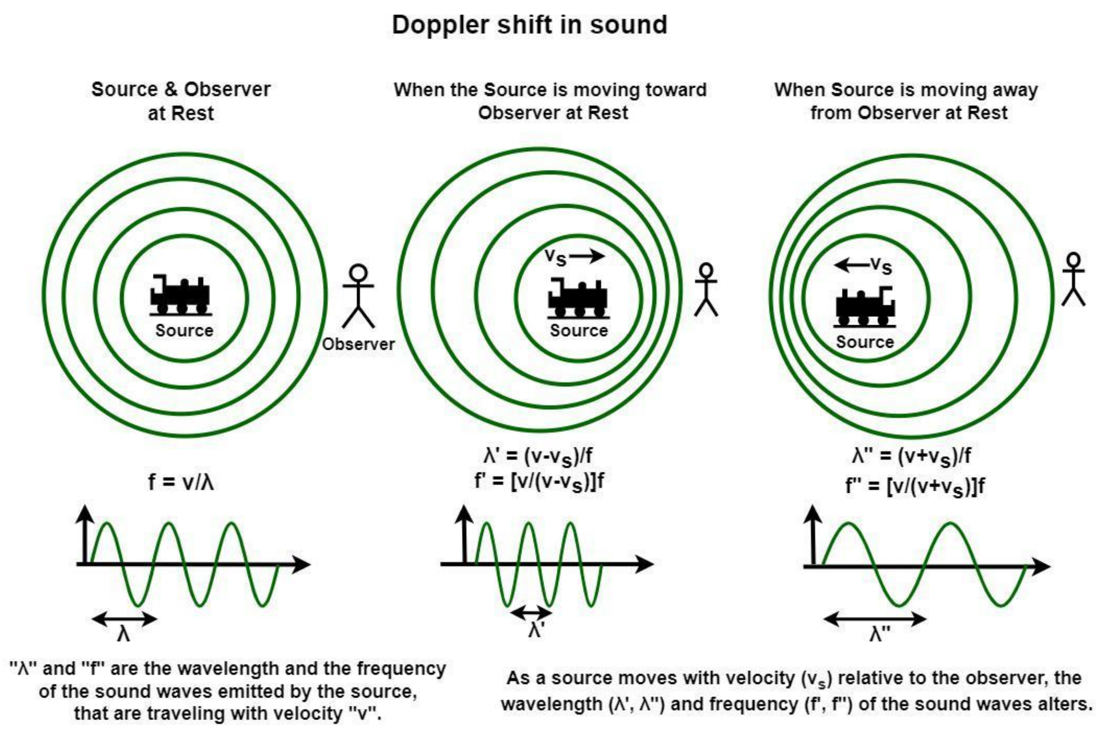

Doppler shift in sound
Source & Observer at Rest
When the Source is moving toward Observer at Rest
When Source is moving away from Observer at Rest
Source
Observer
v_s
Source
v_s
Source
$f = v/\lambda$
$\lambda' = (v-v_S)/f$
$f' = [v/(v-v_S)]f$
$\lambda'' = (v+v_S)/f$
$f'' = [v/(v+v_S)]f$
λ
λ'
λ''
"λ" and "f" are the wavelength and the frequency of the sound waves emitted by the source, that are traveling with velocity "v".
As a source moves with velocity (v_s) relative to the observer, the wavelength (λ', λ") and frequency (f', f") of the sound waves alters.

Figure 4 Doppler effect (Cheng, 2014)

In practice, Doppler affects digital radio systems in several ways. First, the instantaneous carrier frequency perceived by the receiver is shifted, which can break carrier synchronization and degrade demodulation if not tracked. Second, in multicarrier modulations such as OFDM the Doppler spread causes inter-carrier interference (ICI) because different subcarriers experience different phase rotations within a symbol interval — a known problem for high-mobility vehicular channels. Third, the Doppler dynamics limit the useful duration of channel state information: a high relative speed (typical in vehicular scenarios) reduces coherence time and forces the system to use more robust modulation/coding and faster channel tracking. For these reasons your simulation must include Doppler shift calculations per path (angle-dependent) to correctly reproduce time-varying fading and to evaluate receiver algorithms under realistic mobility.

4.1.3 TIME-VARIANT NATURE OF THE V2V CHANNEL

A key characteristic that differentiates vehicular communications from conventional urban wireless links is the high temporal variability of the channel during transmission. In typical fixed or low-mobility urban communications—such as between a stationary base station and a pedestrian—the channel response can be considered quasi-static over the duration of a packet, meaning that the amplitude and phase of each path remain relatively stable during that interval. This stability allows channel estimation performed at the start of a packet to remain valid until its end.

In contrast, Vehicle-to-Vehicle (V2V) links involve both transmitter and receiver moving at potentially high and independent speeds, often in complex urban environments with multiple reflecting and scattering objects that may also be in motion. As a result, the channel's impulse response can change significantly within just a few milliseconds, even during the transmission of a single OFDM symbol. This phenomenon is a direct consequence of:

- High relative velocities between vehicles, which increase Doppler shifts and reduce channel coherence time.
- Dynamic multipath geometry, as the positions of vehicles, buildings, and other reflectors relative to the communication path change rapidly.
- Short-lived scattering conditions, where some reflected or diffracted paths may appear or disappear abruptly due to occlusions or changes in relative positioning.

The rapid time variation of the channel poses a substantial challenge for equalization and channel estimation in DSRC systems. Conventional estimation techniques, which rely on pilot symbols spaced in time and frequency, can quickly become outdated in high-mobility V2V scenarios, resulting in interpolation errors and degraded symbol detection. This is why advanced equalization strategies—capable of tracking the channel more frequently and with greater accuracy—are critical to maintaining reliable performance in vehicular networks.

This property of the V2V channel represents the fundamental difference between regular urban communications and vehicular communications: while both may experience multipath and Doppler effects, only the latter combines them with such rapid channel variation that the channel state may become outdated within the timespan of a single data frame. The mitigation of these effects is therefore a central focus of the physical layer design for V2V systems, and it directly motivates the need for improved equalization methods described in Section 4.3.

In summary, multipath propagation, the Doppler effect, and the inherently time-variant nature of the V2V channel are three of the most critical challenges in establishing reliable vehicular communications in urban environments. Multipath causes fluctuations in signal strength and potential inter-symbol interference due to the superposition of delayed signal components, while Doppler introduces frequency shifts and rapid channel variations that complicate synchronization and demodulation. In V2V scenarios, these effects are further aggravated by the fact that the channel response can change significantly within the duration of a single transmission frame, due to the high relative speeds and constantly evolving multipath geometry. This level of temporal variability is what fundamentally differentiates vehicular communications from conventional urban wireless links and makes channel estimation and distortion mitigation particularly challenging. Understanding these phenomena is therefore essential for accurately modelling the communication channel, designing robust transmission protocols, and developing receiver algorithms capable of maintaining performance under high mobility and dense urban conditions. By incorporating realistic models of these effects into the simulation, this project ensures that the proposed system can be evaluated under conditions that closely resemble real-world deployments.

4.2 ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING (OFDM)

Orthogonal Frequency Division Multiplexing (OFDM) is a multi-carrier modulation technique that divides the available transmission bandwidth into many closely spaced subcarriers, each carrying a separate low-rate data stream. The subcarriers are mathematically orthogonal, which allows them to overlap in frequency without interfering with one another. This property enables high spectral efficiency and strong resilience to multipath fading and inter-symbol interference (ISI), both of which are prevalent in mobile wireless environments.

In OFDM, data bits are first mapped to modulation symbols (e.g., BPSK, QPSK, QAM) and assigned to their respective subcarriers in the frequency domain. The Inverse Fast Fourier Transform (IFFT) converts these frequency-domain values into a composite time-domain waveform for transmission. At the receiver, the Fast Fourier Transform (FFT) performs the reverse operation, recovering the individual subcarriers. To protect against ISI caused by delayed multipath components, a cyclic prefix (CP) is added to each symbol, acting as a guard interval.

A subset of the subcarriers is reserved for pilots—symbols whose values are known to both transmitter and receiver. These pilots allow the receiver to estimate the channel's frequency response and correct for its effects during equalization. This channel estimation process is critical in environments with high mobility, as the channel may vary significantly from one OFDM symbol to the next.

OFDM is particularly relevant in the context of vehicular communications because it is the modulation scheme adopted by IEEE 802.11p, the physical layer standard used in Dedicated Short-Range Communications (DSRC). In the following section, we examine how DSRC employs OFDM to enable reliable, low-latency communication between vehicles in dynamic road environments.

4.3 DSRC IN V2V COMMUNICATIONS

Building upon the understanding of the propagation challenges described in Section 4.1, it becomes essential to examine the communication technologies specifically designed to operate under such conditions. Vehicular environments demand systems capable of maintaining reliable links despite multipath fading, Doppler shifts, and rapidly changing network topologies. Dedicated Short-Range Communications (DSRC), standardized under IEEE 802.11p, was developed with these exact challenges in mind, offering a robust physical and MAC layer architecture optimized for high-mobility scenarios. In the following subsections, the principles, parameters, and operational flow of DSRC in the context of Vehicle-to-Vehicle (V2V) communications are presented in detail.

Dedicated Short-Range Communications (DSRC) is a wireless communication technology specifically designed to enable reliable, low-latency exchanges of information between vehicles and roadside infrastructure. Standardized under IEEE 802.11p, DSRC operates in the 5.85–5.925 GHz frequency band, using a 10 MHz channel bandwidth. Its primary goal is to support safety-critical applications in Intelligent Transportation Systems (ITS), such as collision avoidance, cooperative adaptive cruise control, and emergency vehicle warnings.

At its core, DSRC is an adaptation of Wi-Fi technology optimized for high-speed vehicular environments. Unlike traditional Wi-Fi, which assumes relatively static nodes, DSRC is designed to cope with rapid topology changes, high mobility, and challenging propagation conditions. It offers latency on the order of milliseconds, enabling vehicles to exchange Basic Safety Messages (BSMs) multiple times per second to continuously update surrounding vehicles about their position, speed, and other dynamic parameters.

The physical layer of DSRC follows the IEEE 802.11p specification, which modifies the OFDM (Orthogonal Frequency Division Multiplexing) parameters of IEEE 802.11a to better suit vehicular channels. The structure of an IEEE 802.11p OFDM frame is shown in Figure 5. It is nearly identical to IEEE 802.11a, except that the symbol duration is doubled, improving robustness against multipath. Each packet begins with a preamble containing Short Training Symbols (STSs) and Long Training Symbols (LTSs) for synchronization and channel estimation, followed by a Signal Field carrying information such as modulation type, coding rate, and frame length, and finally the Data Section containing the payload. The FFT size is 64 points, with a subset of subcarriers used as pilots for tracking the channel phase over time. IEEE 802.11p supports data rates from 3 to 27 Mbps, depending on the modulation and coding scheme.

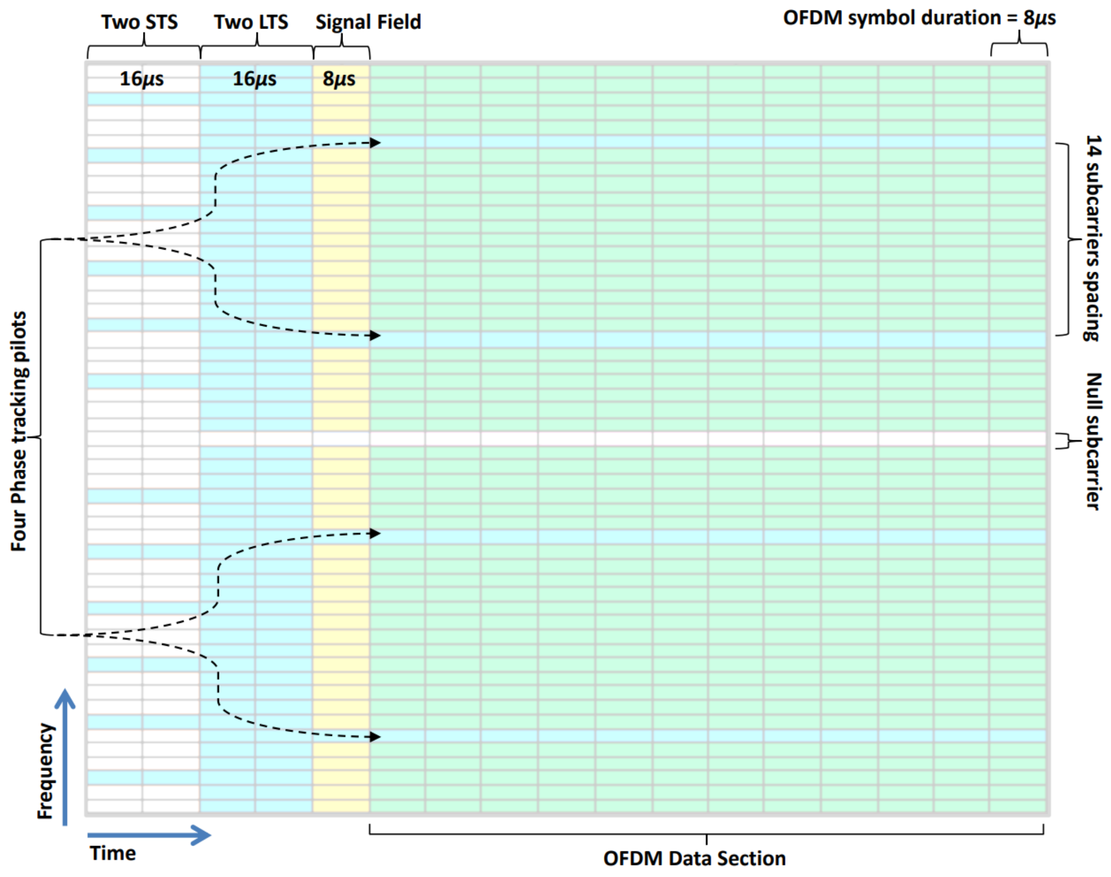
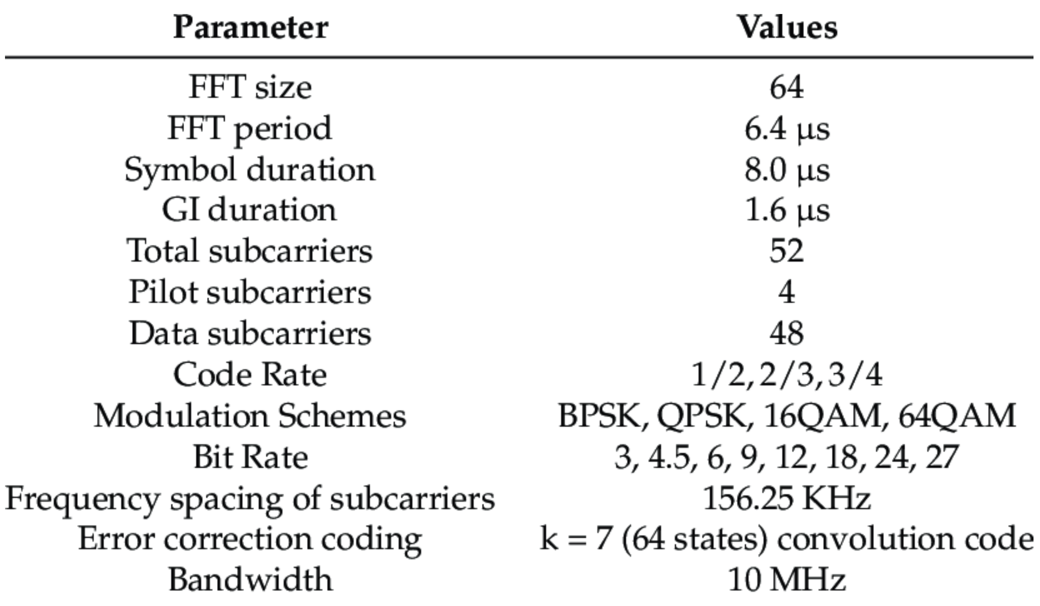


Figure 5. OFDM Frame format of the IEEE 802.11p (Kenney, 2020)

The main parameters of the DSRC physical layer are summarized in Figure 6, which lists the FFT size, subcarrier allocation, symbol duration, guard interval (GI), modulation schemes, coding rates, and bit rates. These settings are chosen to balance robustness against multipath distortion with spectral efficiency. For instance, the reduced channel bandwidth (10 MHz instead of Wi-Fi's typical 20 MHz) doubles the OFDM symbol duration, making the system more resistant to inter-symbol interference caused by multipath in urban environments. The supported modulation schemes range from robust BPSK to higher-order 64QAM, allowing adaptive modulation based on channel quality.



Parameter	Values
FFT size	64
FFT period	6.4 µs
Symbol duration	8.0 µs
GI duration	1.6 µs
Total subcarriers	52
Pilot subcarriers	4
Data subcarriers	48
Code Rate	1/2, 2/3, 3/4
Modulation Schemes	BPSK, QPSK, 16QAM, 64QAM
Bit Rate	3, 4.5, 6, 9, 12, 18, 24, 27
Frequency spacing of subcarriers	156.25 KHz
Error correction coding	k = 7 (64 states) convolution code
Bandwidth	10 MHz

Figure 6 Main DSRC physical layer parameters (Kenney, 2020)

The end-to-end DSRC communication process is shown in Figure 7, which presents the transmitter and receiver block diagrams. On the transmit side, input data bits are first scrambled to avoid long sequences of identical bits. Forward Error Correction (FEC) encoding is then applied using a convolutional code (with a constraint length k=7) to improve resilience against transmission errors. The encoded bits are interleaved to mitigate the impact of burst errors and mapped to modulation symbols (BPSK, QPSK, 16QAM, or 64QAM). These symbols are then modulated onto multiple subcarriers using the Inverse Fast Fourier Transform (IFFT), and a Guard Interval (GI) is added to prevent inter-symbol interference from multipath echoes.

At the receiver, the process is reversed: after detecting the packet, the GI is removed, and a Fast Fourier Transform (FFT) is performed to recover the frequency-domain symbols. Frequency offset correction is applied to compensate for Doppler shifts, followed by demodulation, de-interleaving, and FEC decoding. Finally, the bitstream is descrambled to obtain the original transmitted data.

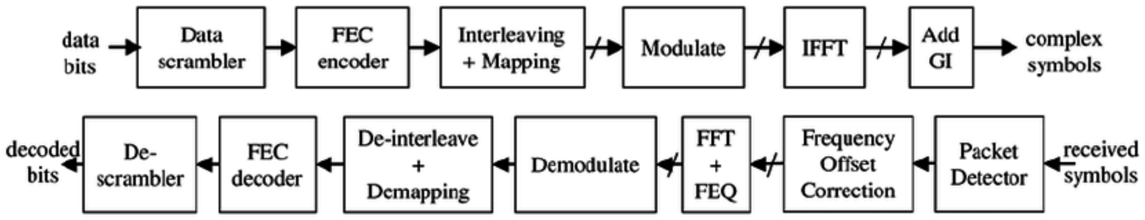


Figure 7 DSRC transmitter and receiver block diagram (AutoTalks, n.d.)

The main advantage of DSRC is its ability to provide direct, infrastructure-free communication between vehicles with extremely low latency, making it ideal for safety applications that require immediate reaction. Its standardized design ensures interoperability across manufacturers, and its use of a reduced bandwidth channel improves resilience to multipath in urban environments. While DSRC's operational range and supported speeds are lower compared to some more modern solutions such as C-V2X, these differences are not critical in scenarios where the communication objective is straightforward and does not require complex networking or wide-area coverage. In such cases—such as this project, where the sole purpose is to broadcast an ambulance's position to nearby vehicles—DSRC offers a simpler, more direct, and fully adequate solution without the additional complexity and overhead of cellular-based alternatives.

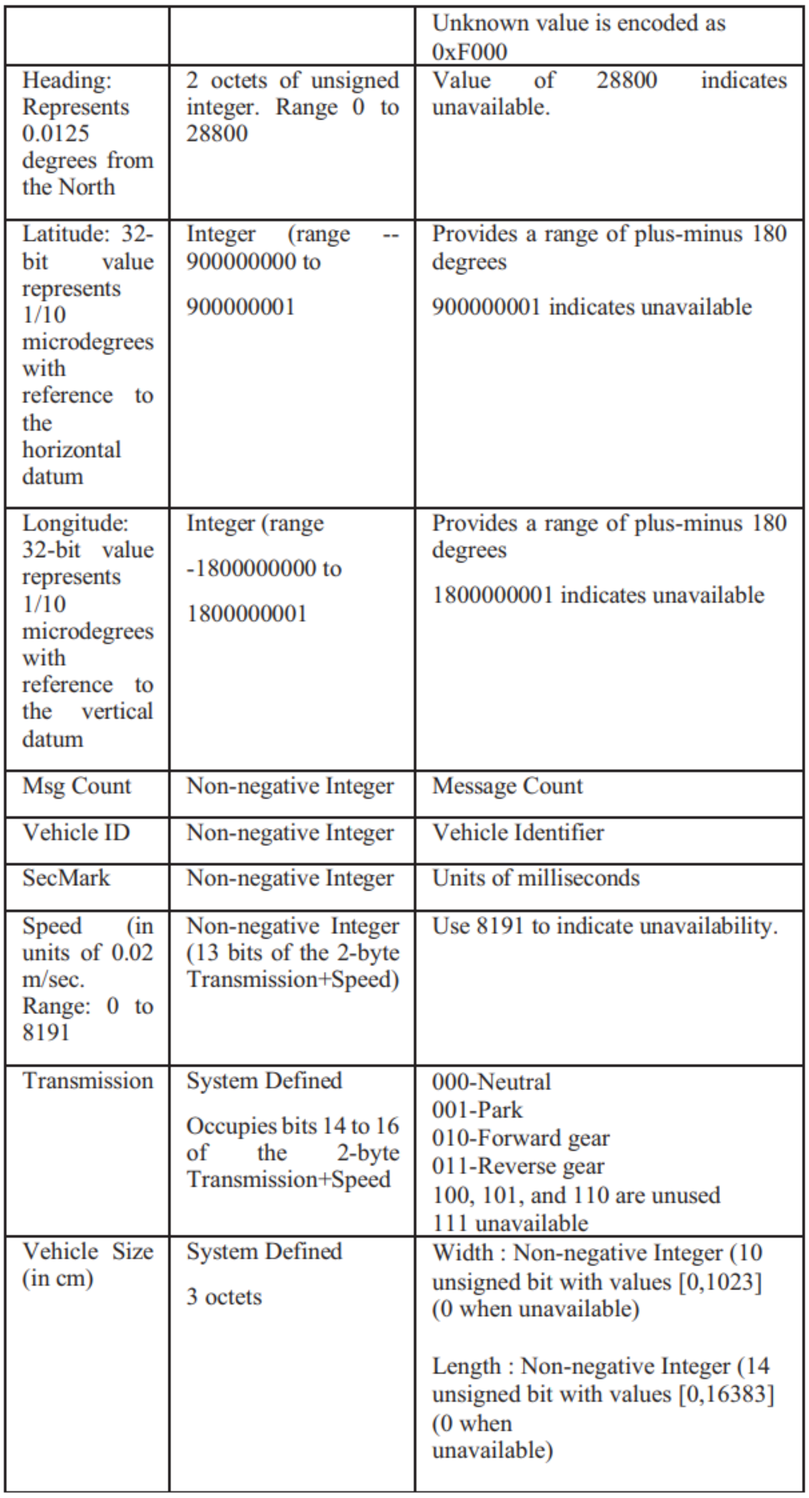
In the context of this project, DSRC serves as the reference standard for the physical and MAC layers, particularly regarding the encoding and transmission of BSMs. By adhering to IEEE 802.11p parameters and architecture, the simulation can realistically evaluate performance under urban vehicular conditions and ensure compatibility with existing V2V communication frameworks.

4.4 BASIC SAFETY MESSAGES (BSM)

As introduced in the previous section, Dedicated Short-Range Communications (DSRC) enables vehicles to exchange Basic Safety Messages (BSMs) to enhance situational awareness and support safety-critical applications. BSMs are defined by the SAE J2735 standard and are the primary broadcast packets used in V2V communication for conveying real-time information about a vehicle's state. Each equipped vehicle periodically transmits BSMs—typically several times per second—to inform surrounding vehicles and infrastructure of its position, motion, and other status parameters.

A BSM consists of two parts: Part I, known as the *BSMcoreData*, and Part II, which is optional and can carry application-specific or extended information. The *BSMcoreData* is mandatory and contains the fundamental fields required for most V2V safety functions. These include identifiers, precise latitude and longitude, elevation, speed, heading, acceleration components, brake status, and vehicle dimensions, among others. The structure and encoding of these fields are standardized to ensure interoperability between different manufacturers and systems. Part II can be used to include additional data such as vehicle path history, path prediction, or specialized safety information, depending on the application requirements. All the fields from the *BSMcoreData* are listed in the tables below:

Data Item	Data Type	Description
Acceleration	System Defined (in units of 0.01 m/sec^2	acceleration_set_4_way : long accel: integer, (acceleration along the X-axis or the direction of travel; negative value indicates braking action) lat accel: integer, (acceleration along the Y-axis or the direction of travel; negative value indicates left turning action, positive indicates right turning) vert accel: one-byte signed integer, (-127 represents unavailable data) yaw: integer (vehicle rotation about the vertical axis and expressed in 0.01 degrees/second)
Angle: Steering Wheel Angle (units of 1.5 degree)	Signed Integer (range: -189 to +189)	0x01 = 00 = +1.5 degree 0x81 = -126 = -189 degree and beyond 0x7E = +126 = +189 degree and beyond 0x7F = +127 to be used for unavailable
Brake System Status: 2-octet information about the current brake system of the vehicle	System Defined	brakeAppliedStatus; (4 bits total--one bit for each wheel, value 1 means active; Thus, 0000 means all Off, 0001 left front active, 0010 left rear active, 0100 right front active, 1000 right rear active) brakesUnavailableStatus: (5th bit; 1 means true) sparebit: 6th bit unused; set to 0 traction: (7th and 8th bits) (Traction Control Status) 00-unavailable) 01-off 10-on but not engaged 11-engaged, abs: (9th and 10th bits) (Anti-lock Brake Status) 00-unavailable; 01-off; 10-on but not engaged; 11-engaged. Stability Control Status, Brake Boost Applied, and Auxiliary/Emergency Brake Status omitted for brevity.
Elevation (unit is 10 cm)	Integer	Elevations from 0 to 61439 (0x0000 to 0xEFFF) meters Elevations from -409.5 to -0.1 (0xF001 to 0xFFF) meters



		Unknown value is encoded as 0xF000
Heading: Represents 0.0125 degrees from the North	2 octets of unsigned integer. Range 0 to 28800	Value of 28800 indicates unavailable.
Latitude: 32-bit value represents 1/10 microdegrees with reference to the horizontal datum	Integer (range --900000000 to 900000001	Provides a range of plus-minus 180 degrees 900000001 indicates unavailable
Longitude: 32-bit value represents 1/10 microdegrees with reference to the vertical datum	Integer (range -1800000000 to 1800000001	Provides a range of plus-minus 180 degrees 1800000001 indicates unavailable
Msg Count	Non-negative Integer	Message Count
Vehicle ID	Non-negative Integer	Vehicle Identifier
SecMark	Non-negative Integer	Units of milliseconds
Speed (in units of 0.02 m/sec. Range: 0 to 8191	Non-negative Integer (13 bits of the 2-byte Transmission+Speed)	Use 8191 to indicate unavailability.
Transmission	System Defined Occupies bits 14 to 16 of the 2-byte Transmission+Speed	000-Neutral 001-Park 010-Forward gear 011-Reverse gear 100, 101, and 110 are unused 111 unavailable
Vehicle Size (in cm)	System Defined 3 octets	Width : Non-negative Integer (10 unsigned bit with values [0,1023] (0 when unavailable) Length : Non-negative Integer (14 unsigned bit with values [0,16383] (0 when unavailable)

Figure 8. BSM Core Data (Guillermo Francia, 2023)

For the purposes of this project, the information contained within the *BSMcoreData* is more than sufficient. In particular, the latitude and longitude fields provide the essential geolocation data needed to determine the exact position of the ambulance. When combined with speed and heading, this positional data allows for accurate calculation of direction and distance relative to other vehicles. This makes it possible to update nearby drivers with timely and precise situational information, ensuring they can respond appropriately in emergency scenarios.

The usefulness of BSMs in this context lies in their standardization, compact size, and high transmission frequency, which together ensure that position updates are both reliable and immediate. By leveraging the existing DSRC framework and its mandatory core data elements, this project can achieve its objective—broadcasting the ambulance's location to surrounding vehicles—without the need for additional, more complex message structures. This not only simplifies implementation but also guarantees compatibility with any DSRC-equipped vehicle adhering to the SAE J2735 standard.

4.5 EQUALIZER IN DSRC

As described in Section 4.2, DSRC employs an OFDM-based physical layer to cope with the harsh propagation conditions present in vehicular environments. However, even with the robustness of OFDM, the transmitted symbols are still affected by multipath fading and Doppler shifts, as discussed in Section 4.1. These impairments distort the amplitude and phase of each subcarrier, degrading the accuracy of symbol detection at the receiver. To mitigate these effects, DSRC systems incorporate an equalizer block in the receiver chain, tasked with estimating the channel response and compensating for its distortions before demodulation.

In the IEEE 802.11p transceiver architecture (see Figure 7), the equalization process is part of the Frequency Offset Correction / FFT / FEQ stage. After converting the received signal into the frequency domain, the equalizer applies a complex gain to each subcarrier to reverse the channel's effect, ideally restoring the transmitted constellation points. This process relies on pilot subcarriers (known symbols inserted at predefined positions) to estimate the channel frequency response and interpolate it for the data subcarriers.

Several equalization algorithms can be employed in DSRC receivers, each with different trade-offs between complexity, robustness, and accuracy:

- Zero Forcing (ZF): A straightforward method that directly inverts the estimated channel response for each subcarrier. While simple to implement, it amplifies noise in subcarriers where the channel gain is low, potentially degrading performance in deep fades.
- Minimum Mean Square Error (MMSE): Improves upon ZF by balancing channel inversion with noise suppression, yielding better performance in low SNR conditions at the cost of slightly higher computational complexity.
- Least Squares (LS): Estimates the channel based solely on pilot subcarriers without assuming prior knowledge of the channel or noise statistics. It is simple but less accurate in noisy or highly time-varying channels.
- Decision-Directed (DD): Uses detected data symbols, in addition to pilots, to refine channel estimates iteratively. This method can improve accuracy in slowly varying channels but may propagate errors if the initial decisions are incorrect.

The choice of equalization method has a direct impact on system performance in fast-changing vehicular channels. In particular, urban V2V communications are characterized by both severe multipath and significant Doppler spreads, which require equalizers to rapidly adapt to time-varying channel conditions. Standard pilot-based equalizers can struggle in these environments due to the rapid decorrelation of the channel between pilot positions, leading to inaccurate interpolation for data subcarriers.

This challenge motivates the use of more advanced pilot-aided techniques designed specifically for high-mobility scenarios. Among these, the Constructed Data Pilots (CDP) approach has emerged as a highly effective solution for V2V communications in dense urban settings, offering superior channel tracking and distortion compensation. The CDP algorithm is discussed in detail in Section 4.4.

4.6 CONSTRUCTED DATA PILOTS ALGORITHM

4.6.1 CDP ALGORITHM EXPLAINED

The Constructed Data Pilots (CDP) algorithm is a channel estimation enhancement technique for OFDM-based vehicular communications that exploits the correlation between the channel responses of consecutive OFDM symbols. In V2V scenarios, although the channel can vary rapidly, the changes between adjacent symbols are often gradual enough to be predicted and tracked. CDP takes advantage of this property by generating additional “virtual” pilots from the detected data symbols of the current frame and using them to refine the channel estimate for the next symbol. By iteratively updating the estimate in this way, the algorithm produces a channel estimation that progressively follows the true channel evolution. This enables improved robustness in highly dynamic conditions without increasing pilot overhead. The process is illustrated in Figure 9.

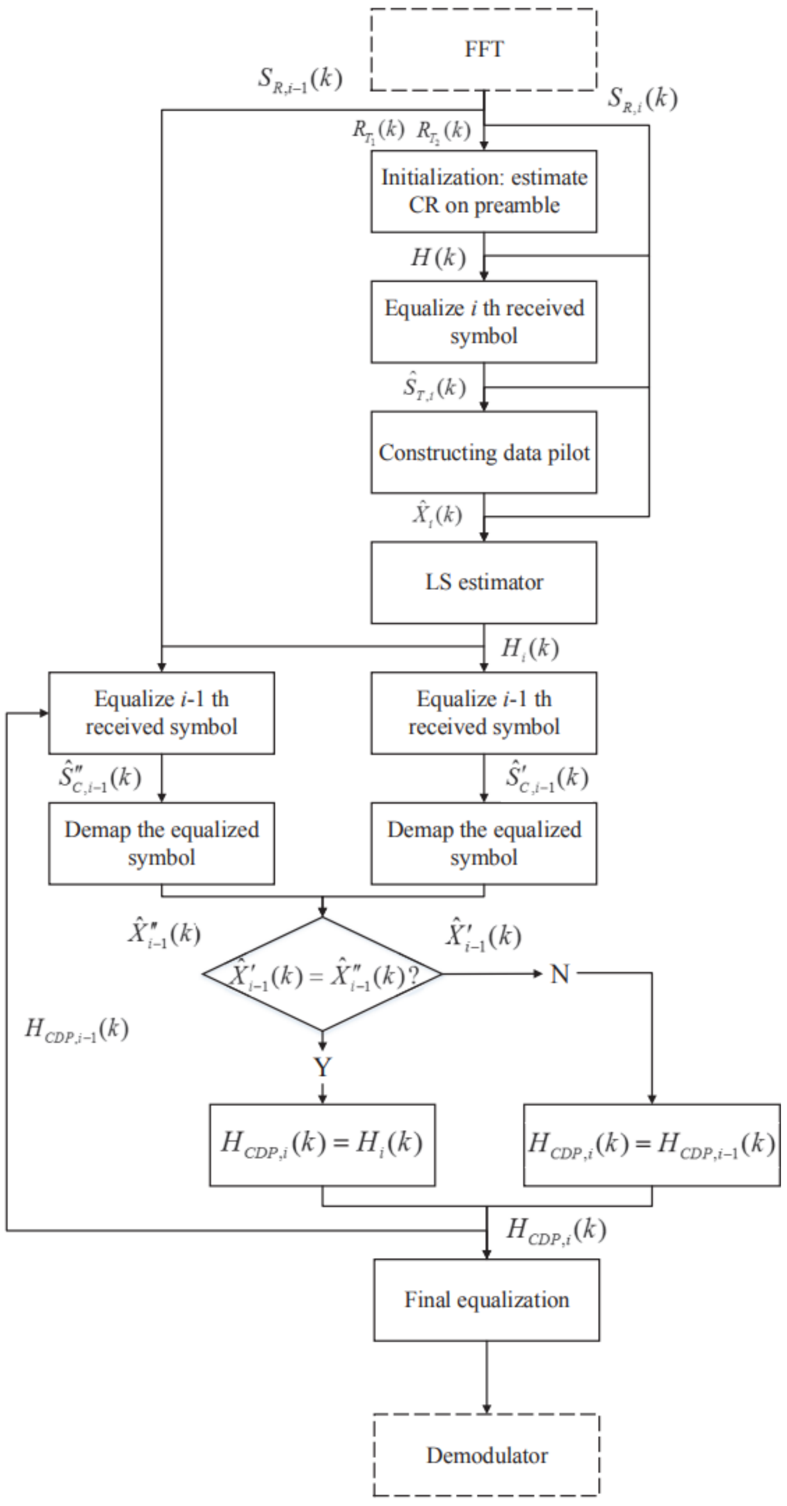

FFT
$S_{R,i-1}(k)$
$S_{R,i}(k)$
$R_{T_1}(k)$ $R_{T_2}(k)$
Initialization: estimate CR on preamble
$H(k)$
Equalize *i* th received symbol
$\hat{S}_{T,i}(k)$
Constructing data pilot
$\hat{X}_i(k)$
LS estimator
$H_i(k)$
Equalize *i*-1 th received symbol
Equalize *i*-1 th received symbol
$\hat{S}''_{C,i-1}(k)$
$\hat{S}'_{C,i-1}(k)$
Demap the equalized symbol
Demap the equalized symbol
$\hat{X}''_{i-1}(k)$
$\hat{X}'_{i-1}(k)$
$\hat{X}'_{i-1}(k)=\hat{X}''_{i-1}(k)?$
N
Y
$H_{CDP,i-1}(k)$
$H_{CDP,i}(k)=H_i(k)$
$H_{CDP,i}(k)=H_{CDP,i-1}(k)$
$H_{CDP,i}(k)$
Final equalization
Demodulator

Figure 9 CDP Algorithm Diagram (Tong Wang, 2018)

The operation begins with the first received OFDM symbol. After the FFT stage, the receiver estimates the channel at the pilot subcarriers using Least Squares (LS) estimation:

$$H_{P,1}(k) = \frac{S_{R,1}(k)}{X_P(k)}$$

Where $S_{R,1}(k)$ is the received frequency-domain symbol and $X_P(k)$ is the known pilot symbol. Using this initial channel estimate, the receiver equalizes the first symbol and demodulates it to obtain the transmitted data bits. These bits are then re-modulated to their corresponding constellation points, producing the constructed data pilots $\hat{X}_{D,1}(k)$ for the data subcarrier positions.

The constructed data pilots are combined with the original pilots and fed into the LS estimator to obtain an updated channel estimate $H_{CDP,1}(k)$ for the entire set of subcarriers. This refined estimate is then used for processing the next OFDM symbol.

For each subsequent symbol i, the equalization is performed using the previous refined estimate $H_{CDP,i-1}(k)$. The detected data is demodulated, re-modulated to generate $\hat{X}_{D,i}(k)$, and combined with the original pilots to update the channel estimate $H_{CDP,i}(k)$. By repeating this process for every symbol, the CDP algorithm continually refreshes the channel estimation based on the most recently received data, allowing the system to better follow rapid channel variations.

As shown in Figure 9, this iterative approach ensures that each OFDM symbol benefits from a channel estimate derived from both fixed pilots and the most recent detected data, increasing robustness to fast time variation while avoiding the need for extra pilot subcarriers. This makes CDP particularly well suited for high-mobility vehicular communication scenarios.

4.6.2 CDP ALGORITHM SIMPLIFIED EXAMPLE

Since the CDP algorithm is quite complex and difficult to visualize due to its recursive nature, it is best to run through a highly simplified example to fully understand how it works in a step-by-step manner. In this example, we are simulating the transmission of 3 symbols, to keep the explanation simple. The 3 symbols that are being transmitted are as follows:

- $S1_T = +1$. This is the pilot symbol (known to the receiver).
- $S2_T = -1$. Unknown to the receiver
- $S3_T = +1$. Unknown to the receiver

Thus, the receiver will receive these 3 symbols, one after the other. It is important to note that the first symbol is a Pilot symbol, which means that the receiver knows exactly what it should be receiving for the first symbol. If the received $S1_R$ is different from the actual $S1_T$, then the receiver knows that the channel is introducing distortion.

The true channel response, for each transmitted symbol, is as follows:

- $H1 = 0{,}4$
- $H2 = 0{,}2$
- $H3 = 0{,}1$

With this in mind, a complete run through the algorithm for the 3 symbols would look like this:

- $S1_T$ (the Pilot symbol) is transmitted.
- $S1_R$ is received at the receiver. $S1_R = S1_T * H1 = 0{,}4$
- CDP estimates the channel: $H1_{est} = \frac{S1_R}{S1_T} = 0{,}4$
- $S2_T$ is transmitted

- $S2_R$ is received. $S2_R = S2_T * H2 = -0{,}2$
- $S2_R$ is equalized to obtain $X2 = \frac{S2_R}{H1_{est}} = -0{,}5$
- $X2$ is rounded to nearest value to obtain $S2_{est} = -1$
- Now, $S2_{est}$ is used as the new Pilot symbol.
- New channel estimation: $H2_{est} = \frac{S2_R}{S2_{est}} = 0{,}2$
- $S3_T$ is transmitted
- $S3_R$ is received. $S3_R = S3_T * H3 = 0{,}1$
- $S3_R$ is equalized to obtain $X3 = \frac{S3_R}{H2_{est}} = 0{,}5$
- $X3$ is rounded to nearest value to obtain $S3_{est} = +1$
- Now, $S3_{est}$ is used as the new Pilot symbol

  New channel estimation: $H3_{est} = \frac{S3_R}{S3_{est}} = 0{,}1$

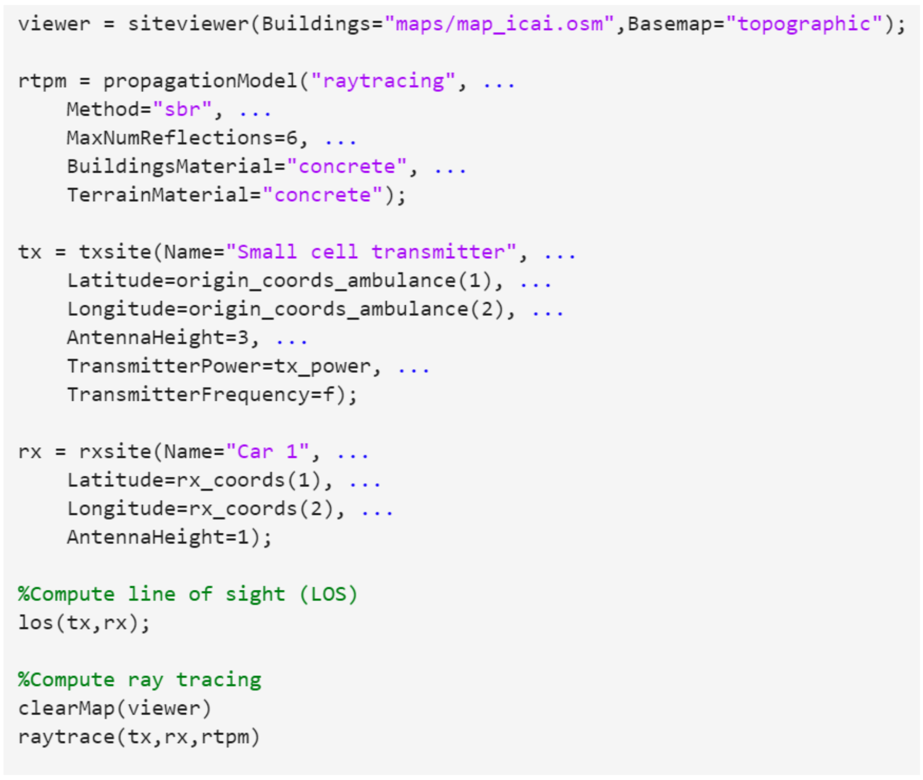
This process would continue with all following symbols. As can be seen, the 3 symbols have been properly estimated but, more importantly, the channel estimations are accurate. This means that the algorithm is correctly following the channel variation, which is a key feature in the context of vehicular communications. If the LS estimation technique had been used for all 3 symbols, although they would have been correctly estimated, the channel response calculated by the model would not be accurate to the real one. Then, this would certainly cause errors in the long run, when more symbols are received, and after the channel has suffered more variations.

Chapter 5. DESIGNED SYSTEM

5.1 INITIAL SETUP

After having studied the current technologies in the context of vehicular communications (Phase 0), it is time to start working on the end goal of this project. Firstly, the objective is to find a way of simulating the transmission of data inside an urban scenario with moving vehicles and simulated obstacles that generate the typical multipath distortions. In order to achieve this, and after a lot of research, everything is going to be programmed in MATLAB, thanks to its strong Communication and Vehicular Networks Toolboxes.

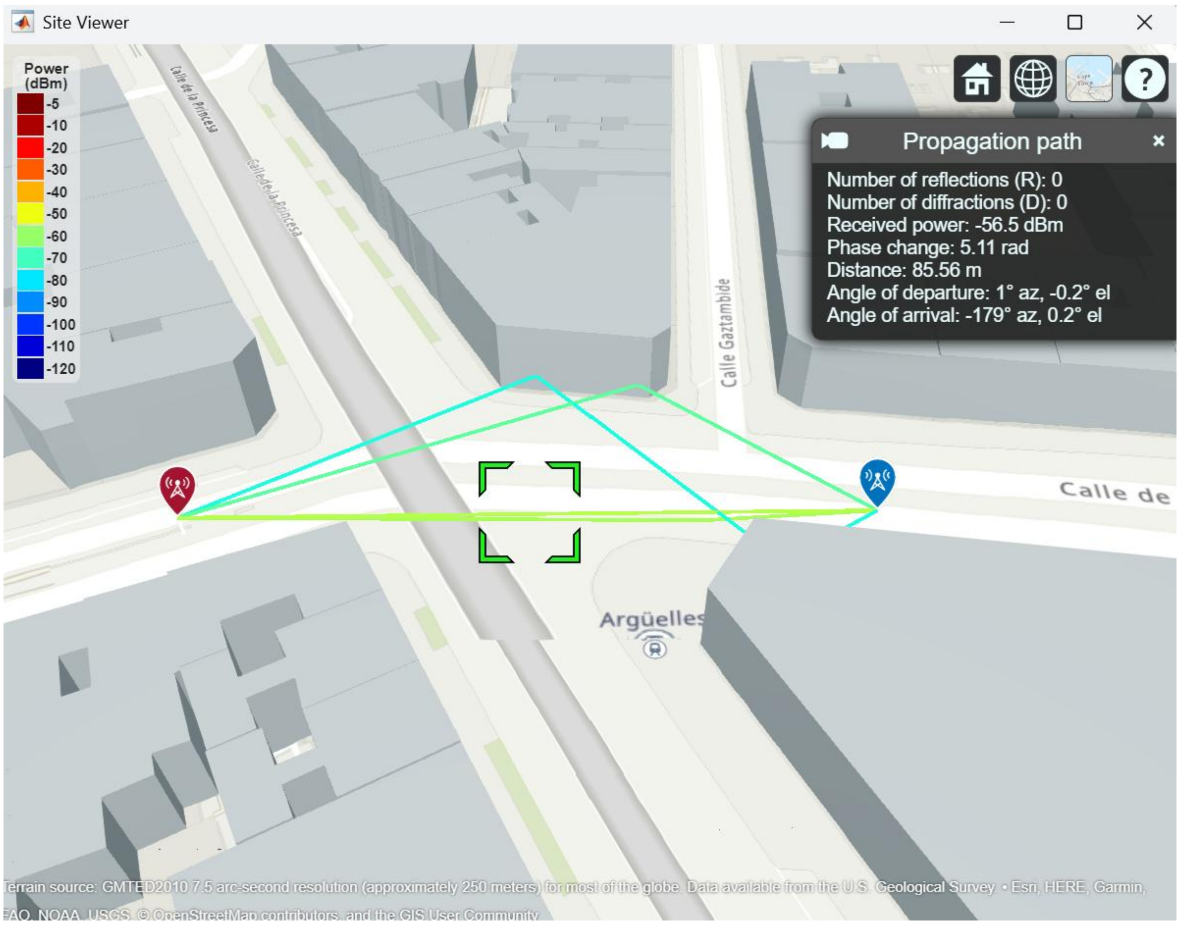
The first step is to find a visual way of creating the necessary scenario inside MATLAB. The method used must provide a simulated urban scenario, with the ability to position transmitters and receivers. All of this can be achieved by using the *siteviewer* function in MATLAB, included in the Communications Toolboxes. Here is an example that shows how to use this function:



```
viewer = siteviewer(Buildings="maps/map_icai.osm",Basemap="topographic");

rtpm = propagationModel("raytracing", ...
    Method="sbr", ...
    MaxNumReflections=6, ...
    BuildingsMaterial="concrete", ...
    TerrainMaterial="concrete");

tx = txsite(Name="Small cell transmitter", ...
    Latitude=origin_coords_ambulance(1), ...
    Longitude=origin_coords_ambulance(2), ...
    AntennaHeight=3, ...
    TransmitterPower=tx_power, ...
    TransmitterFrequency=f);

rx = rxsite(Name="Car 1", ...
    Latitude=rx_coords(1), ...
    Longitude=rx_coords(2), ...
    AntennaHeight=1);

%Compute line of sight (LOS)
los(tx,rx);

%Compute ray tracing
clearMap(viewer)
raytrace(tx,rx,rtpm)
```

Figure 10. SiteViewer example

As can be seen, an OSM file (Open Street Map) must be provided, which can be downloaded online, with the desired area of simulation. For this example, the area selected is an intersection near the center of Madrid, in Spain. Then, the propagation model is created, by establishing the simulation method (raytracing), the maximum number of reflections calculated, the material used for the simulated buildings, etc. Finally, the transmitter (tx) and receiver (rx) are created and positioned inside the scenario. Note that, at this moment, they are both static. After running the code, which computes the line of sight and then calculates the raytracing propagation, this is the obtained result:


Site Viewer
Power (dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Calle de la Princesa
Calle Gaztambide
Calle de
Argüelles
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey • Esri, HERE, Garmin, FAO, NOAA, USGS, © OpenStreetMap contributors, and the GIS User Community

Propagation path
Number of reflections (R): 0
Number of diffractions (D): 0
Received power: -56.5 dBm
Phase change: 5.11 rad
Distance: 85.56 m
Angle of departure: 1° az, -0.2° el
Angle of arrival: -179° az, 0.2° el

Figure 11. SiteViewer Example Result

As shown in Figure 11, the different propagation paths of the transmitted signal are displayed between the transmitter (red) and the receiver (blue), with realistic bouncing physics and power losses. In addition, the program also provides all necessary information about each propagation ray, such as the distance of travel, the angle of departure and arrival, or the received power at the receiver, all of which can be saved and read through MATLAB code. This program establishes a solid ground for this project and is going to be used for all following simulations, growing in complexity.

At this point, it is important to understand what is already being done, and what is missing. What is already being done is:

- Simulated urban environment with buildings made of concrete

- Simulated static transmitter and receiver which can be positioned anywhere inside the scenario.
- Simulated signal propagation with rays bouncing off buildings, creating multiple propagation paths.
- Acquisition of realistic parameters such as received power, link distance, etc.

What is missing is:

- Real transmission of bits using DSRC with OFDM
- Moving transmitter and receiver.
- Applying the simulated multipath losses to the transmission to generate realistic distortion.
- Noise implementation
- Many other features

5.2 SYSTEM DESIGN

Having a simulated scenario, with a realistic channel response, it is time to start designing the real DSRC system that uses OFDM technology to transmit the coordinates of a moving ambulance, following the BSM message structure. For this, a good approach is to follow the block diagram shown in Figure 7, and program each block one by one. For the sake of simplicity, the Scrambler and Frequency Offset Correction block are being removed from the system.

Therefore, the system that needs to be programmed is represented entirely by the following Figure 12

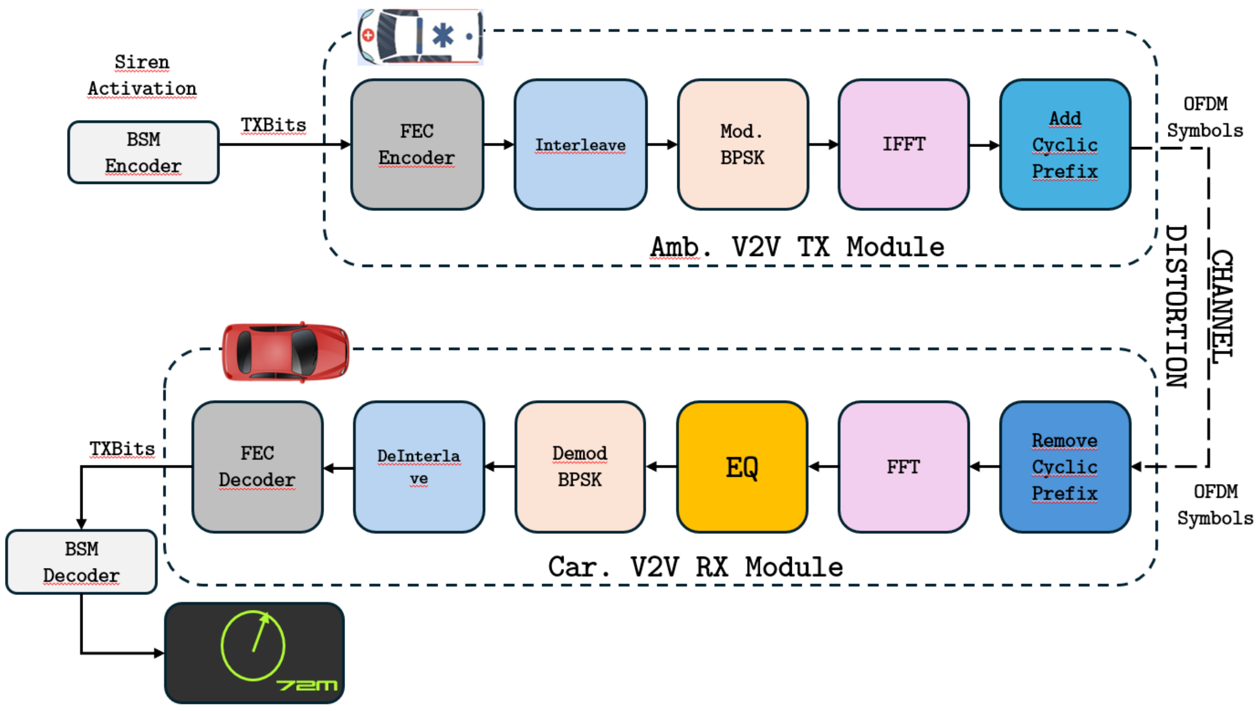


Figure 12. System Block Diagram

In summary, this is what the system does, step by step:

- The ambulance activates the sirens and initiates the system
- Ambulance coordinates enter the BSM Encoder and are converted into a sequence of bits
- These bits are encoded using FEC Convolutional encoding, obtaining a new sequence of bits.
- These bits are interleaved to minimize burst-type errors.
- These bits are then modulated with BPSK and converted into OFDM symbols in the frequency domain
- These symbols are moved back to the time domain using the IFFT
- A cyclic prefix is added to each symbol in the time domain
- The OFDM symbols are transmitted, traversing the channel that generates distortion

- The OFDM symbols are received at the car's receiver
- The Cyclic Prefix is removed from the OFDM symbols
- The symbols are shifted to the frequency domain using an FFT operation
- The symbols go through the Equalizer block (EQ), that eliminates the distortion caused by the channel
- The symbols are demodulated and demapped with BPSK, to obtain the received sequence of bits
- These bits are deinterleaved
- These bits are decoded using the FEC Convolution decoder, obtaining the final sequence of bits.
- This sequence of bits is then decoded by the BSM encoder, that extracts the coordinates of the ambulance.
- Finally, the coordinates of the ambulance are displayed to the driver of the car in a visual manner.

With this process in mind, it is time to start programming each block.

5.2.1 BSM ENCODER BLOCK

The BSM Encoder block is responsible for generating the binary representation of a Basic Safety Message (BSM), which is the core data packet transmitted in this V2V communication system. BSMs follow the SAE J2735 standard, and in this project, the *coreData* portion is used to carry essential vehicle status information.

Although a complete BSM contains many fields—including position, speed, heading, acceleration, brake status, and vehicle size—this implementation focuses primarily on the latitude and longitude fields, as these are sufficient to meet the project's objective: broadcasting the position of the ambulance to surrounding vehicles. The remaining fields are populated with fixed "mock" values, ensuring that the message structure remains compliant while avoiding unnecessary complexity.

The MATLAB implementation is shown below:

```
function bsm_bits = generateBSM(lat_deg, long_deg)%GENERATEBSM Summary of this
function goes here


    % Function to generate a BSM coreData message in bits
    % Inputs:
    %   lat_deg  - latitude in degrees (WGS-84)
    %   long_deg - longitude in degrees (WGS-84)
    %
    % Output:
    %   bsm_bits - row vector of bits representing the BSM

    % Helper function to convert an integer to bit array
    to_bits = @(value, bits) de2bi(value, bits, 'left-msb');

    % Example field values
    msgCnt = 1;                           % 8 bits
    id = hex2dec('12345678');            % 32 bits
    secMark = 50000;                     % 16 bits (ms)
    lat = round(lat_deg * 1e7);          % 1/10 microdegrees
    long = round(long_deg * 1e7);        % same scale
    elev = round(100 / 0.1);             % 16 bits, 0.1m resolution (e.g., 100m)
    semi_major = 200;                    % 8 bits, 0.05m resolution
    semi_minor = 200;                    % same
    transmission = 2;                    % 4 bits (e.g., forward gears)
    speed = round(15 / 0.02);            % 13 bits, 0.02 m/s resolution (e.g., 15
m/s)
    heading = round(90 / 0.0125);        % 16 bits, 0.0125 degrees (e.g., 90°)
    angle = round(10 / 1.5);             % 8 bits, signed (-189 to 189°)
    accel_long = round(1 / 0.01);        % 10 bits, 0.01 m/s² resolution
    accel_lat = round(0 / 0.01);         % 10 bits
    accel_vert = 127;                    % 8 bits, -6.4 to +6.35 m/s²
    yaw_rate = round(0 / 0.0125);        % 16 bits, 0.0125 deg/s
    brake_flags = [0 0 0 0];             % 4 bits, placeholder
    brake_pressure = 15;                 % 4 bits, max
    vehicle_length = 500;                % 12 bits, in cm (e.g., 5 m)
    vehicle_width = 180;                 % 9 bits, in cm (e.g., 1.8 m)

    lat_bits = to_bits(typecast(int32(lat), 'uint32'), 32);
    long_bits = to_bits(typecast(int32(long), 'uint32'), 32);

    % Convert each field to bit arrays
    bsm_bits = [
        to_bits(msgCnt, 8), ...
        to_bits(id, 32), ...
        to_bits(secMark, 16), ...
        lat_bits, ...
        long_bits, ...
        to_bits(typecast(int16(elev), 'uint16'), 16), ...
        to_bits(semi_major, 8), ...
        to_bits(semi_minor, 8), ...
        to_bits(transmission, 4), ...
        to_bits(speed, 13), ...
        to_bits(heading, 16), ...
        to_bits(typecast(int8(angle), 'uint8'), 8), ...
        to_bits(typecast(int16(accel_long), 'uint16'), 10), ...
        to_bits(typecast(int16(accel_lat), 'uint16'), 10), ...
        to_bits(accel_vert, 8), ...
        to_bits(yaw_rate, 16), ...
        brake_flags, ...
        to_bits(brake_pressure, 4), ...
        to_bits(vehicle_length, 12), ...
        to_bits(vehicle_width, 9)
    ];
end
```

The function accepts the vehicle's position in degrees, lat_deg and long_deg, and converts these into the integer-scaled representation required by the SAE J2735 standard. This scaling corresponds to units of 0.1 microdegrees, providing sub-meter positional accuracy when decoded.

The to_bits helper function is used throughout to convert numeric values into binary arrays of fixed length, using big-endian (left-most bit as the most significant bit) ordering.

The message fields are then assembled in the exact order specified by the standard. Some examples include:

- Message Count (msgCnt, 8 bits) – an incremental counter to track consecutive BSMs.
- Temporary ID (id, 32 bits) – a unique identifier for the transmitting vehicle.
- Time (secMark, 16 bits) – the millisecond mark of the message within the current minute.
- Position (lat, long, 32 bits each) – the vehicle's latitude and longitude, scaled and stored as signed integers.
- Speed (speed, 13 bits) – vehicle speed in increments of 0.02 m/s.

- Heading (heading, 16 bits) – direction of travel in increments of 0.0125 degrees.

Other fields such as acceleration components, brake status, and vehicle dimensions are included with placeholder values to maintain message format compatibility. This ensures that the generated message could, in principle, be understood by any standard-compliant BSM decoder. Finally, all field bit arrays are concatenated into a single row vector.

The result is bsm_bits, a complete binary representation of the BSM ready for transmission through the encoding, modulation, and OFDM framing stages of the system.

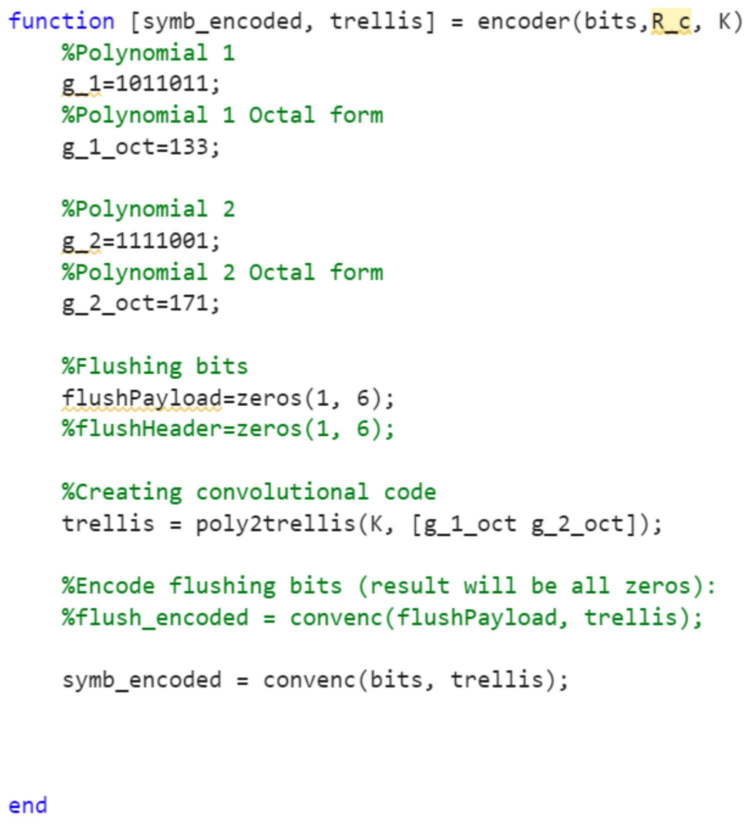
The BSM Encoder serves as the data source for the entire transmitter chain. By encoding the vehicle's geographic location into the standard BSM format, it ensures that the transmitted information can be understood by any DSRC-equipped receiver adhering to SAE J2735. In the context of this project, the latitude and longitude fields provide the critical real-time position of the ambulance, enabling nearby vehicles to react appropriately. The other fields, although not used in decision-making here, maintain compliance and interoperability with the wider V2V communication ecosystem.

5.2.2 FEC ENCODER / DECODER BLOCKS

The Forward Error Correction (FEC) stage is responsible for adding controlled redundancy to the transmitted bitstream so that the receiver can detect and correct a certain number of errors introduced by the channel without requiring retransmission. In the implemented system, this is achieved using a rate-1/2 convolutional code with constraint length K=7 and two generator polynomials defined by the IEEE 802.11p standard.

5.2.2.1 FEC Encoder

At the transmitter (Amb. V2V TX Module in Figure 12), the input bit sequence generated by the BSM encoder is passed through the convolutional encoder implemented in MATLAB. The relevant code is shown below:



```
function [symb_encoded, trellis] = encoder(bits,R_c, K)
    %Polynomial 1
    g_1=1011011;
    %Polynomial 1 Octal form
    g_1_oct=133;

    %Polynomial 2
    g_2=1111001;
    %Polynomial 2 Octal form
    g_2_oct=171;

    %Flushing bits
    flushPayload=zeros(1, 6);
    %flushHeader=zeros(1, 6);

    %Creating convolutional code
    trellis = poly2trellis(K, [g_1_oct g_2_oct]);

    %Encode flushing bits (result will be all zeros):
    %flush_encoded = convenc(flushPayload, trellis);

    symb_encoded = convenc(bits, trellis);


end
```

Figure 13. FEC Encoder code

The encoder uses the poly2trellis function to generate the trellis structure from the generator polynomials [133oct,171oct]. These correspond to the standard polynomials used in 802.11p and many other communication systems, optimized for error correction performance. The constraint length K=7 means the encoder has six memory elements, allowing it to consider the current input bit and the six previous bits when producing each output. The coding rate Rc=1/2 indicates that for each input bit, the encoder generates two output bits—one from each generator polynomial—thereby doubling the bit rate at this stage and introducing the redundancy needed for error correction.

5.2.2.2 FEC Decoder

At the receiver (Car. V2V RX Module in Figure 12), the encoded bitstream is decoded using a Viterbi algorithm, which performs maximum-likelihood sequence estimation to recover the most probable transmitted bit sequence given the received noisy sequence. The implemented decoder uses hard-decision decoding, where the demodulator outputs binary values before decoding. The decoding process is carried out with the following line of MATLAB code:

```
rx_bits_all = vitdec(rx_bits_all, trellis, app.tbl, 'trunc', 'hard');
```

Here, vitdec applies the Viterbi algorithm to the received bits rx_bits_all using the same trellis structure generated at the encoder. The parameter app.tbl specifies the traceback depth, which determines how far back in the received sequence the algorithm looks when making a decision on a bit. The 'trunc' option is used to process independent blocks of data without maintaining state between them, which is appropriate for packet-based communication. The 'hard' setting specifies that the input bits are binary (0 or 1), rather than soft values or likelihood ratios.

By combining convolutional encoding at the transmitter with Viterbi decoding at the receiver, the system gains resilience to bit errors caused by noise, fading, and interference in the wireless channel. This ensures that critical data, such as the Basic Safety Messages transmitted by the ambulance, can be recovered accurately even under challenging vehicular communication conditions.

5.2.3 INTERLEAVER / DEINTERLEAVER BLOCKS

The interleaving process is an important step in digital communication systems, designed to mitigate the effects of burst errors introduced by the wireless channel. Burst errors occur when several consecutive bits are corrupted, which can overwhelm the error-correcting capability of the FEC decoder if left unaddressed. Interleaving combats this by reordering the encoded bit sequence in a predictable way before transmission, so that consecutive bits in the original sequence are spread out in time and frequency over the transmitted signal. At the receiver, the inverse process—deinterleaving—is applied to restore the original ordering of bits before FEC decoding.

In the implemented system (Figure 12), the Interleave block is placed immediately after the FEC encoder in the transmitter chain, while the DeInterleave block is placed immediately before the FEC decoder in the receiver chain. This ensures that the redundancy introduced by the FEC is preserved and optimally utilized after the bits are reordered.

The MATLAB implementation of the interleaver is shown below:

```
% ONLY WORKS WITH ROW VECTORS
function matrix_output = interleave(matrix, rows, columns)
    % Converts vector into a matrix with dimension rows x columns
    matrix_interm = reshape(matrix, rows, columns);
    % Transpose matrix to organize bits
    matrix_final = matrix_interm';
    % Back to vector
    matrix_output = matrix_final(:)';
end
```

This interleaver takes the encoded bit sequence and reshapes it into a 2D matrix with the specified number of rows and columns. A matrix transpose is then applied, which rearranges the ordering of bits in a structured, deterministic way. Finally, the transposed matrix is converted back into a row vector for further processing. The exact choice of rows and columns controls the interleaving depth, which should be set according to the expected error burst length in the channel.

The corresponding deinterleaver, placed in the receiver chain, simply reverses this process.

In this V2V communication system, the interleaver/deinterleaver pair plays a key role in protecting Basic Safety Messages (BSMs) against channel impairments. In the presence of frequency-selective fading or impulsive noise—common in vehicular environments—errors tend to occur in bursts. Without interleaving, such bursts could corrupt contiguous bits belonging to the same FEC codeword, reducing the likelihood of successful error correction. By dispersing the bits across time and frequency, interleaving ensures that any burst errors affect bits from different codewords, allowing the FEC decoder to recover the original message more reliably.

5.2.4 MODULATION BLOCK (BPSK AND OFDM)

The Modulation block in the transmitter is responsible for converting the interleaved binary data stream into complex symbols suitable for transmission over the channel. In this system, Binary Phase Shift Keying (BPSK) is used as the modulation scheme. BPSK is one of the most robust digital modulation methods, representing each bit as a symbol located at one of two points on the constellation diagram, 180° apart in phase. A logical '0' is mapped to −1 and a logical '1' is mapped to +1 in the real axis of the complex plane. This simplicity makes BPSK highly resilient to noise and an appropriate choice for safety-critical vehicular communications.

When visualized on a constellation diagram (see Figure 14), the modulated symbols occupy two points along the real axis—−1 and +1—corresponding to logical '0' and '1', respectively. This makes detection straightforward and robust, which is especially valuable in the high-mobility and interference-prone conditions of vehicular networks.

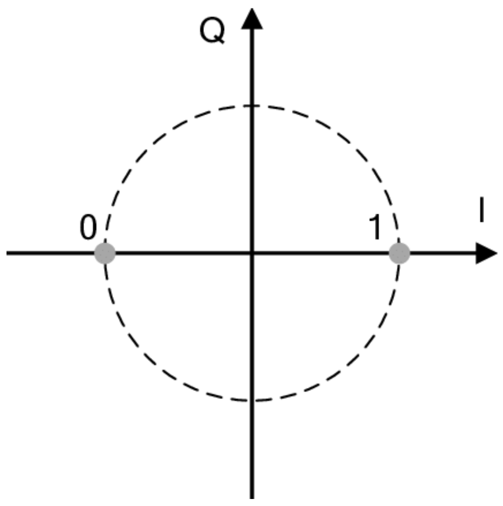


Figure 14. BPSK Constellation Diagram (Mathworks, s.f.)

The MATLAB implementation begins by reshaping the bitstream into groups corresponding to the number of data subcarriers per OFDM symbol, nBitsSymbBPSK. This reshaping ensures that each OFDM symbol contains exactly the number of modulated symbols required for the data subcarriers:

```
% Map to BPSK for this BSM
X_blocks_data = 2 * reshape(txbits, app.nBitsSymbBPSK, nSym_per_BSM) - 1;
```

Here, the factor of 2 and subtraction of 1 perform the binary-to-BPSK mapping

The result is a matrix where each column represents the BPSK symbols for the data subcarriers of one OFDM symbol.

Next, the system constructs complete OFDM blocks by inserting both the modulated data symbols and the pilot symbols into their respective subcarrier positions:

```
% Build OFDM blocks
X_blocks = zeros(app.NFFT, nSym_per_BSM);
for s = 1:nSym_per_BSM
    X_blocks(app.data_indexes, s) = X_blocks_data(:, s);
    X_blocks(app.pilot_indexes, :) = 1;
End
```

The variable app.NFFT specifies the FFT size, which determines the total number of subcarriers in the OFDM symbol. Data symbols are placed at the indices specified by app.data_indexes, while pilot subcarriers are set to a constant value of +1+1+1. These pilot tones are known at the receiver and are used for channel estimation and phase tracking.

By the end of this stage, each column of X_blocks represents one complete OFDM symbol in the frequency domain, containing both data and pilot subcarriers. These frequency-domain symbols are then ready for the IFFT block, which will transform them into the time-domain signal for transmission.

5.2.5 IFFT BLOCK

The IFFT block is responsible for converting the modulated OFDM symbols from the frequency domain into the time-domain waveform that will be transmitted over the channel. In Orthogonal Frequency Division Multiplexing (OFDM), each subcarrier carries an independently modulated symbol (either a data symbol or a pilot symbol). In the frequency domain, these subcarriers are represented as discrete samples of the symbol spectrum. To transmit them as a continuous waveform, they must be converted to their corresponding time-domain representation using the Inverse Fast Fourier Transform (IFFT).

The IFFT operation efficiently performs the following transformation:

$$x_{td}[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k] \cdot e^{j2\pi kn/N}$$

Where:

- $X[k]$ are the complex-valued frequency-domain samples for the k-th subcarrier.
- N is the FFT size (app.NFFT).
- $x_{td}[n]$ are the resulting time-domain samples for $n = 0, 1, \dots, N-1$.

In the MATLAB implementation, the transformation is carried out with a single command:

```
x_td = ifft(X_block, app.NFFT);  % IFFT to time domain
```

Here, X_block contains one complete OFDM symbol in the frequency domain, with both data and pilot subcarriers already placed in their respective positions. The function ifft transforms this symbol into x_td, the corresponding time-domain waveform.

The use of the IFFT in OFDM ensures that all subcarriers remain orthogonal to each other in the time domain, even though they overlap in frequency. This orthogonality is the key to OFDM's high spectral efficiency—it allows the subcarriers to be spaced closely without causing inter-carrier interference.

The output of the IFFT block is a sequence of complex-valued samples representing the composite OFDM symbol in the time domain. These samples are then passed to the Cyclic

Prefix Addition block, which prepares them for transmission over a multipath channel by improving their robustness to inter-symbol interference (ISI).

5.2.6 CYCLIC PREFIX ADDITION BLOCK

The Cyclic Prefix (CP) Addition block is the final step in the OFDM symbol preparation before transmission. Its main purpose is to combat inter-symbol interference (ISI) and preserve the orthogonality of the subcarriers in the presence of multipath propagation—an especially common phenomenon in vehicular environments where reflections from buildings, vehicles, and roadside objects are prevalent.

In multipath channels, delayed copies of the transmitted signal can overlap with the following symbol, causing ISI. The cyclic prefix mitigates this by extending each OFDM symbol with a guard interval formed by copying the last Ncp samples of the time-domain symbol and placing them at the beginning. This ensures that the linear convolution of the transmitted signal with the channel's impulse response appears as a circular convolution to the receiver's FFT process, thus maintaining subcarrier orthogonality.

The MATLAB implementation is straightforward:

```
x_cp = [x_td(end-app.Ncp+1:end); x_td];  % Add CP: append last Ncp samples
```

By inserting the cyclic prefix, any multipath components arriving within the guard interval duration will not interfere with the orthogonal detection of the subcarriers. Although this process slightly reduces spectral efficiency—since the cyclic prefix contains redundant data—it significantly improves robustness to delay spread, making it a standard feature in OFDM-based systems like IEEE 802.11p.

Once the cyclic prefix has been added, the OFDM symbol is ready for transmission through the wireless channel, where it will be subject to noise, fading, and Doppler effects before being processed by the receiver chain.

5.2.7 Cyclic Prefix removal block

On the receiver side, the Remove Cyclic Prefix block is the first step in reconstructing the OFDM symbols from the received time-domain waveform. Its role is to discard the guard interval that was added at the transmitter, leaving only the original N-sample OFDM symbol for demodulation.

When the cyclic prefix was added at the transmitter, the last Ncp samples of the OFDM symbol were prepended to the front of the symbol to create a guard interval. This guard interval absorbs the effects of multipath propagation as long as the maximum delay spread of the channel is shorter than the cyclic prefix length. At the receiver, these extra samples are not part of the actual OFDM data and must be removed before applying the FFT.

The MATLAB code for this operation is:

```
y_td_no_cp = y_td(app.Ncp+1 : app.Ncp+app.NFFT);  % Remove CP: take NFFT samples
after CP
```

Removing the cyclic prefix restores the correct symbol alignment for the FFT operation. This step is critical to maintain the orthogonality between subcarriers—any misalignment here would cause inter-carrier interference (ICI) and degrade system performance.

Once the cyclic prefix has been removed, the time-domain OFDM symbol y_td_no_cp is ready to be passed to the FFT block, which will transform it back to the frequency domain for subsequent channel estimation and demodulation.

5.2.8 FFT Block

The FFT block is responsible for converting the received OFDM symbol from the time domain back into the frequency domain, where the individual subcarriers can be processed independently. This step is the reverse operation of the IFFT block at the transmitter and is essential for recovering the modulated symbols carried on each subcarrier.

In OFDM, the transmitted signal is constructed such that all subcarriers remain orthogonal in the time domain, allowing them to overlap in frequency without interference. By applying the FFT to the received signal after cyclic prefix removal, the receiver separates these subcarriers into distinct frequency-domain bins, each corresponding to one subcarrier index.

$$Y[k] = \frac{1}{N}\sum_{n=0}^{N-1} y_{td}[n] \cdot e^{-j2\pi kn/N}$$

Where:

- $y_{td}[n]$ is the n-th time-domain sample after cyclic prefix removal.
- N is the FFT size (app.NFFT).
- $Y[k]$ is the complex symbol for the k-th subcarrier.

In MATLAB, this is implemented with a single command:

```
Y_block = fft(y_td_no_cp, app.NFFT);  % FFT to frequency domain
```

The output Y_block contains the complex-valued frequency-domain symbols for all subcarriers, including both data subcarriers (which carry modulated information bits) and pilot subcarriers (which are known reference symbols used for channel estimation and phase tracking).

By transforming the received signal into the frequency domain, the FFT enables the system to handle the effects of the channel on each subcarrier individually. This prepares the data for the subsequent Channel Estimation and Equalization stages, where channel distortions will be compensated before symbol detection.

5.2.9 EQUALIZER BLOCK

The Equalization block is arguably the most critical component of the receiver chain. Its purpose is to compensate for the distortions introduced by the wireless channel, specifically, the frequency-selective fading and phase shifts affecting each subcarrier independently in an OFDM system. Without proper equalization, even a perfectly demodulated OFDM symbol will carry corrupted data, making accurate bit recovery impossible.

In this system, the equalization process is performed using an improved Constructed Data Pilots (iCDP) algorithm. This method goes beyond simple Least Squares (LS) estimation by exploiting the correlation between consecutive OFDM symbols to gradually refine the channel estimate. The approach is particularly well-suited to highly dynamic vehicular channels, where rapid channel variation can cause conventional pilot-only estimators to lag behind actual conditions.

The MATLAB call to the equalization function is:

```
[X_hat_eq, H_est_prev, remod_prev, Y_block_prev] = ...
    equalization_2(X_block, Y_block, 'icdp', app.symbol_book, ...
    app.n_taps, H_est_prev, remod_prev, Y_block_prev, 1:app.NFFT);
```

- X_block contains the transmitted pilot symbols for reference,
- Y_block is the received frequency-domain OFDM symbol (output from the FFT block),
- app.symbol_book defines the modulation constellation (BPSK in this case),
- H_est_prev, remod_prev, and Y_block_prev store results from the previous symbol for iterative refinement.

The function equalization_2 is defined here:

```
function [X_hat_eq, H_est_out, remod_out, y_fd_prev] = equalization_2(X_block,
X_hat_block, ch_est_method, symbol_book, n_taps, H_est_prev, remod_prev,
Y_block_prev, data_indexes, pilot_indexes)
% Performs equalization using iCDP on selected data subcarriers only

NFFT = length(X_block);
X_hat_eq = zeros(NFFT,1);  % Initialize full vector

if nargin < 6, H_est_prev = []; end
if nargin < 7, remod_prev = []; end
if nargin < 8, Y_block_prev = []; end
if nargin < 9, data_indexes = [6:10, 12:24, 26:31, 33:38, 40:52, 54:58]; end
if nargin < 10, pilot_indexes = [11, 25, 39, 53]; end

switch lower(ch_est_method)
    case 'icdp'
        if isempty(H_est_prev) || isempty(remod_prev)
            % First symbol: direct channel estimation
            H_est = zeros(NFFT,1);
            H_est(data_indexes) = X_hat_block(data_indexes) ./
X_block(data_indexes);

            X_hat_eq(data_indexes) = X_hat_block(data_indexes) ./
H_est(data_indexes);
            rec_syms = knnsearch([real(symbol_book), imag(symbol_book)], ...
                                 [real(X_hat_eq(data_indexes)),
imag(X_hat_eq(data_indexes))]) - 1;
            remod = symbol_book(rec_syms + 1);

            H_est_out = H_est;
            remod_out = zeros(NFFT,1);
            remod_out(data_indexes) = remod;
            remod_out(pilot_indexes) = 1;
            y_fd_prev = X_hat_block;

        else
            % Step 1: Equalize current symbol with H_prev
            X_hat_eq_1 = zeros(NFFT,1);
            X_hat_eq_1(data_indexes) = X_hat_block(data_indexes) ./
H_est_prev(data_indexes);

            rec_syms = knnsearch([real(symbol_book), imag(symbol_book)], ...
                                 [real(X_hat_eq_1(data_indexes)),
imag(X_hat_eq_1(data_indexes))]) - 1;
            remod = symbol_book(rec_syms + 1);

            % Step 2: Estimate new H from current symbol
            H_est_new = zeros(NFFT,1);
            H_est_new(data_indexes) = X_hat_block(data_indexes) ./ remod;

            % Step 3: Consistency check on PREVIOUS symbol
            X_prev_check = zeros(NFFT,1);
            X_prev_check(data_indexes) = Y_block_prev(data_indexes) ./
H_est_new(data_indexes);

            rec_syms_check = knnsearch([real(symbol_book), imag(symbol_book)],
...
                                       [real(X_prev_check(data_indexes)),
imag(X_prev_check(data_indexes))]) - 1;
            remod_check = symbol_book(rec_syms_check + 1);

            % Step 4: Accept new estimate if majority matches
            match_count = sum(remod_check(data_indexes) ==
remod_prev(data_indexes));
            %match_count = sum(remod_check == remod_prev(data_indexes));
            match_ratio = match_count / length(data_indexes);


            if match_ratio >= 0.9
                H_est_refined = H_est_new;
                % Check if pilots have been reversed --> check for phase
                % correction
                n_pilots_reversed = sum(remod(pilot_indexes) ~=
X_block(pilot_indexes));
                if n_pilots_reversed == length(pilot_indexes)
                    H_est_refined = H_est_refined * exp(-1j * pi);
                end
            else
                H_est_refined = H_est_prev;
            end



            % Step 5: Final equalization with selected channel
            X_hat_eq(data_indexes) = X_hat_block(data_indexes) ./
H_est_refined(data_indexes);
            H_est_out = H_est_refined;

            remod_out = zeros(NFFT,1);
            remod_out(data_indexes) = remod;

            y_fd_prev = X_hat_block;
        end
    otherwise
        error('Only iCDP implemented in equalization_2');
end
end
```

This code mimics the exact logic explained in Section 4.6, but with a few improvements. The iCDP algorithm operates in two modes:

1. Initialization (first symbol) – Since there is no prior channel estimate, the algorithm performs a direct per-subcarrier LS estimation using the known pilots and data symbol positions. It then demodulates the data subcarriers to the nearest constellation points and remodulates them to form "constructed" data pilots. These constructed pilots are stored for use in the next iteration.

2. Improved Refinement (subsequent symbols) – For each new OFDM symbol, the previous channel estimate is used to perform a first equalization pass. The equalized symbols are detected, remodulated, and used to produce a fresh channel estimate. To ensure stability, the new estimate is validated by checking its consistency against the remodulated symbols from the previous iteration. Only if the match ratio exceeds a set threshold (90% in this implementation) is the refined estimate accepted.

The algorithm also includes a phase reversal check for pilot subcarriers, correcting for possible π-phase flips caused by certain channel conditions.

By reusing detected data symbols as additional "virtual" pilots, iCDP increases the effective pilot density without increasing pilot overhead. This allows the channel estimate to better follow rapid variations, improving equalization accuracy and ultimately reducing the bit error rate in challenging V2V environments.

The output of this block, X_hat_eq, contains the equalized frequency-domain symbols for the data subcarriers, ready to be demapped back into bits in the subsequent demodulation stage. The updated channel estimate (H_est_prev), remodulated symbols (remod_prev), and last received block (Y_block_prev) are passed forward to maintain the iterative process for the next OFDM symbol.

In summary, the Equalization block is the point at which the system directly counteracts the channel impairments that would otherwise make reliable communication impossible. The iCDP method's ability to track fast channel variations is essential for maintaining the robustness and reliability needed in safety-critical V2V applications, such as broadcasting the position of an emergency vehicle in real time.

5.2.10 DEMODULATION BLOCK

The Demodulation block is responsible for converting the equalized frequency-domain symbols back into the original transmitted bits. Since this system uses Binary Phase Shift Keying (BPSK), the decision process is straightforward: each symbol lies on the real axis at either +1 or −1, corresponding to a transmitted bit of '1' or '0', respectively.

The MATLAB implementation is:

```
% Demodulate received symbols for this symbol
demod_bits = real(X_hat_eq) > 0;
demod_bits_all(:, s) = demod_bits;

% Prepare transmitted bits for reference
tx_bits = X_block;
tx_bits(app.data_indexes) = (X_block(app.data_indexes) + 1) / 2;
tx_bits(app.pilot_indexes) = 1;
```

Here:

- X_hat_eq contains the equalized frequency-domain symbols for the current OFDM symbol (output of the Equalization block).
- real(X_hat_eq) > 0 performs a hard-decision demodulation:
  - If the real part is greater than zero → decision = bit '1'.
  - If the real part is less than or equal to zero → decision = bit '0'.
- The resulting demod_bits vector is stored in the demod_bits_all matrix, with each column corresponding to one OFDM symbol in the current Basic Safety Message (BSM) frame.

The second part of the code prepares a tx_bits reference from X_block (the original transmitted frequency-domain block), converting BPSK symbols back into bits using (x+1)/2 for data subcarriers, and assigning pilot subcarriers a constant value of '1'. This reference is useful for performance evaluation, such as calculating the bit error rate (BER) during simulation.

Since BPSK modulation has only two constellation points, the demodulation step is computationally simple yet robust to noise. When combined with the enhanced equalization provided by the iCDP algorithm, it ensures that bit decisions are made with the highest possible accuracy given the prevailing channel conditions.

The output of this block is a sequence of recovered bits for each OFDM symbol, ready to be passed to the Deinterleaving and FEC Decoding blocks. These subsequent stages will restore the original bit order and correct any residual errors before reconstructing the transmitted Basic Safety Message.

5.2.11 BSM DECODER BLOCK

Having already explained the Deinterleaving and FEC Decoding blocks, the last step left to do is to recover the ambulance coordinates from the received sequence of bits. This is done in the BSM Decoder block.

The BSM Decoder block is the final stage in the receiver chain. Its role is to recover the vehicle position information—latitude and longitude—from the binary Basic Safety Message (BSM) received after the complete demodulation, deinterleaving, and decoding process. This step is essential because, while the earlier stages ensure the integrity of the transmitted bits, it is only here that these bits are interpreted as meaningful, application-level data.

The MATLAB implementation is:

```
function [lat_deg, long_deg] = extractLatLongFromBSM(bsm_bits)
```

```
    % Extracts latitude and longitude in degrees from a BSM bit vector
    % Inputs:
    %   bsm_bits - vector of bits representing the BSM
    % Outputs:
    %   lat_deg  - latitude in degrees
    %   long_deg - longitude in degrees

    % Bit ranges (0-indexed logic)
    lat_start = 8 + 32 + 16;        % after msgCnt (8), id (32), secMark (16)
    long_start = lat_start + 32;

    % Extract 32 bits for lat and long
    lat_bits = bsm_bits(lat_start+1 : lat_start+32);
    long_bits = bsm_bits(long_start+1 : long_start+32);

    % Convert from bits to signed integers
    lat_uint32 = bi2de(lat_bits, 'left-msb');
    long_uint32 = bi2de(long_bits, 'left-msb');

    % Typecast back to signed 32-bit
    lat = typecast(uint32(lat_uint32), 'int32');
    long = typecast(uint32(long_uint32), 'int32');

    % Convert to degrees
    lat_deg = double(lat) / 1e7;
    long_deg = double(long) / 1e7;
end
```

The BSM Decoder provides the final usable output of the entire V2V communication chain: the precise geographic coordinates of the transmitting vehicle. In the context of this project, this means determining the real-time position of the ambulance from the BSMs received over the simulated IEEE 802.11p link.

While the earlier blocks ensure that the transmitted bits are faithfully recovered, this block transforms those bits into meaningful application-layer information that can be used for decision-making, such as alerting nearby drivers or triggering automated vehicle responses.

For this project's purpose, where only latitude and longitude are required for the application, the BSM Decoder focuses solely on these fields. All other fields in the BSM remain unused, although they are still present in the bitstream to maintain compliance with the SAE J2735 standard.

5.3 TESTING

With the system fully designed, the next step is to start testing it in a simulated scenario. To achieve this, it is necessary to build a fully working scenario that mimics moving vehicles in an urban intersection. As seen in Section 5.1, the initial setup provides a real intersection, with simulated buildings, and the ability to position the transmitter (ambulance) and receiver (car) anywhere on the map. Although this is a good start, it lacks the most important aspect of vehicular communications: the movement of vehicles. Therefore, at this point, the most important task is to create a testing environment that uses the mentioned initial setup, but that implements all the moving parts.

5.3.1 TESTING SCRIPT

This is achieved by a custom MATLAB script, shown here:

```
% ==== PARAMETERS ====
f = 5.9e9;
tx_power = 1;                % Watts
B = 10e6;                    % Bandwidth Hz
map = 'maps/map_chicago.osm';
encoding = "no";
interleaving = "no";
view_plots = "yes";
view_scene = "yes";
ch_est_method = 'iCDP';      % Per-symbol equalization
simulate_channel_distortion = "yes";
mod_order = 1;               % BPSK
ambulance_speed = 20;        % m/s
car_speed = 20;              % m/s
BSMs_per_second = 100;        % BSMs every 100 ms

% OFDM parameters
nBitsSymbBPSK = 48;
NFFT = 64;
nSubCar = 52;
Ncp = 16;
Tofdm = 8e-6;    % 8 us per OFDM symbol
n_taps = 8;      % Required by equalization fn
pilot_indexes = [11, 25, 39, 53];
data_indexes = [6:10, 12:24, 26:31, 33:38, 40:52, 54:58];
symbol_book = [-1; 1];       % BPSK

% Encoding parameters
if encoding == "yes"
```

```
    R_c = 1/2;
    K=7;
    tbl=32;
else
    R_c = 1;
end
%origin_coords_ambulance = [41.894968, -87.623526];
origin_coords_ambulance = [41.895613,-87.624195];
%origin_coords_ambulance = [41.895572,-87.624363];

%dest_coords_ambulance   = [41.894968, -87.624014];
dest_coords_ambulance = [41.895125,-87.624174];
%dest_coords_ambulance = [41.895164,-87.624342];

%rx_coords   = [41.895423, -87.624358];
rx_coords = [41.895900,-87.624206];
%rx_coords = [41.894927,-87.624906];

%dest_coords_rx = [41.895033, -87.624357];
dest_coords_rx = [41.895434,-87.624190];
%dest_coords_rx = [41.894945,-87.624579];

%origin_coords_ambulance = [41.894968, -87.623526];
origin_coords_ambulance = [41.895764,-87.622897];

%dest_coords_ambulance   = [41.894968, -87.624014];
dest_coords_ambulance = [41.895747,-87.624470];

%rx_coords   = [41.895423, -87.624358];
rx_coords = [41.896486,-87.624390];
%dest_coords_rx = [41.895033, -87.624357];
dest_coords_rx = [41.895709,-87.624373];

bits_per_BSM = 266;
nSym_per_BSM = ceil(bits_per_BSM / nBitsSymbBPSK);

dist_ambulance = haversine(origin_coords_ambulance, dest_coords_ambulance) *
1000;
dist_car = haversine(rx_coords, dest_coords_rx) * 1000;

nBSMs = floor((dist_ambulance / ambulance_speed) * BSMs_per_second);
sim_duration = dist_ambulance / ambulance_speed; % seconds
BSM_period = 1 / BSMs_per_second;

% Generate all positions along trajectories at OFDM-symbol resolution
total_OFDM_symbols = nBSMs * nSym_per_BSM/R_c;
ambulance_lats  = linspace(origin_coords_ambulance(1), dest_coords_ambulance(1),
total_OFDM_symbols);
ambulance_longs = linspace(origin_coords_ambulance(2), dest_coords_ambulance(2),
total_OFDM_symbols);
car_lats  = linspace(rx_coords(1), dest_coords_rx(1), total_OFDM_symbols);
car_longs = linspace(rx_coords(2), dest_coords_rx(2), total_OFDM_symbols);
```

```
% -- Ray Tracing Model Setup --
viewer = siteviewer(Buildings=map, Basemap="topographic");
rtpm = propagationModel("raytracing", ...
    Method="sbr", ...
    MaxNumReflections=6, ...
    BuildingsMaterial="concrete", ...
    TerrainMaterial="concrete");

% ==== MAIN SIMULATION LOOP ====
txbits = [];
BER_vec = zeros(1, nBSMs);


for bsm_idx = 1:nBSMs
    n_errors_BSM = zeros(nSym_per_BSM,1);
    demod_bits_all = zeros(NFFT, nSym_per_BSM);
    tx_bits_all = zeros(NFFT, nSym_per_BSM);
    idx_start = (bsm_idx-1)*nSym_per_BSM + 1;
    idx_end   = min(bsm_idx*nSym_per_BSM, total_OFDM_symbols);

    % Generate BSM with ambulance's position at BSM start
    lat_this_BSM  = ambulance_lats(idx_start);
    long_this_BSM = ambulance_longs(idx_start);
    BSM_bits = generateBSM(lat_this_BSM, long_this_BSM);
    BSM_bits = strrep(num2str(BSM_bits), ' ', '');
    pad_len = nSym_per_BSM * nBitsSymbBPSK - length(BSM_bits);
    BSM_bits = [BSM_bits, repmat('0', 1, pad_len)];
    BSM_bits = BSM_bits - '0';
    %txbits = double(BSM_bits) - '0';
    txbits = BSM_bits;

    % ENCODING
    %Encoding block
    if encoding=="yes"
        [txbits, trellis]=encoder(txbits, R_c, K);
        nSym_per_BSM = nSym_per_BSM/R_c;
    end

    % Map to BPSK for this BSM
    X_blocks_data = 2 * reshape(txbits, nBitsSymbBPSK, nSym_per_BSM) - 1;

    % Build OFDM blocks
    X_blocks = zeros(NFFT, nSym_per_BSM);
    for s = 1:nSym_per_BSM
        X_blocks(data_indexes, s) = X_blocks_data(:, s);
        X_blocks(pilot_indexes, :) = 1;
    end


    % -- Per-symbol transmission and reception --
    H_est_prev = [];
    remod_prev = [];
    Y_block_prev = [];
    rx_symbols_eq = zeros(size(X_blocks));

    for s = 1:nSym_per_BSM
        X_block = X_blocks(:, s);
        idx_global = idx_start + s - 1;
        % Get transmitter and receiver position for this symbol
        tx_lat = ambulance_lats(idx_global);
        tx_long = ambulance_longs(idx_global);
        rx_lat = car_lats(idx_global);
        rx_long = car_longs(idx_global);

        % Setup txsite and rxsite for ray tracing
        tx = txsite(Name="Ambulance", ...
            Latitude=tx_lat, ...
            Longitude=tx_long, ...
            AntennaHeight=3, ...
            TransmitterPower=tx_power, ...
            TransmitterFrequency=f);

        rx = rxsite(Name="Car", ...
            Latitude=rx_lat, ...
            Longitude=rx_long, ...
            AntennaHeight=1);

        if view_scene == "yes"
            clearMap(viewer)
            show(tx)
            %los(tx,rx)
            raytrace(tx,rx,rtpm)
        end

        % Ray trace to get rays (multipath) at this instant
        rays = raytrace(tx, rx, rtpm);
        rays = rays{1};

        % --- Channel impulse response from rays ---
        fs = (NFFT + Ncp) / Tofdm;
        h_n = generate_channel_distortion(rays, fs);


        % -- OFDM Symbol Construction with CP --
        x_td = ifft(X_block, NFFT);                            % IFFT to time domain
        x_cp = [x_td(end-Ncp+1:end); x_td];                    % Add CP: append last
Ncp samples


        % -- Channel: multipath convolution --
        y_td = conv(x_cp, h_n);

        % Convert to frequency domain
        H_true = fft(h_n, NFFT).';
        % Simulate received symbols
        %Y_block = H_true .* X_block;
        %y_td_2 = ifft(Y_block, NFFT);

        % -- Calculate SNR --
        P_rx = sigstrength(rx,tx,rtpm);
        SNR_dB = calculate_SNR(P_rx, B);


        % -- Add AWGN noise --
        data_pwr = mean(abs(y_td).^2);
        noise_pwr = data_pwr / 10^(SNR_dB/10);
        noise = sqrt(noise_pwr/2)*(randn(size(y_td))+1j*randn(size(y_td)));
        %y_td = y_td + noise;

        % -- Remove CP and FFT to frequency domain --
        y_td_no_cp = y_td(Ncp+1:Ncp+NFFT);                 % Remove CP: take NFFT
samples after CP
        Y_block = fft(y_td_no_cp, NFFT);                   % FFT to frequency
domain


         % iCDP equalization
        %fprintf('---- FDM symbol (iCDP) ----\n');
        %[X_hat_eq, H_est, remod] = equalization_2(X_block, Y_block, 'icdp',
symbol_book, n_taps, [], [], [], [1:NFFT]);
        %[X_hat_eq, H_est_prev, remod_prev, Y_block_prev] =
equalization_2(X_block, Y_block, 'icdp', symbol_book, n_taps, H_est_prev,
remod_prev, Y_block_prev, 1:NFFT);
        [X_hat_eq, H_est_prev, remod_prev, Y_block_prev] =
equalization_2(X_block, Y_block, 'icdp', symbol_book, n_taps, H_est_prev,
remod_prev, Y_block_prev, 1:NFFT);


        % -- Demodulate received symbols for this symbol --
        demod_bits = real(X_hat_eq) > 0;
        demod_bits_all(:, s) = demod_bits;
        tx_bits = X_block;
        tx_bits(data_indexes) = (X_block(data_indexes) + 1) / 2;
        tx_bits(pilot_indexes) = 1;
        %tx_bits_all(:, s) = tx_bits;

        n_errors = sum(demod_bits(data_indexes) ~= tx_bits(data_indexes));
        n_errors_BSM(s) = n_errors;
        BER_sym = n_errors / length(tx_bits);

        clf;

        % PLOT CONSTELLATIONS
        if view_plots == "yes"

            %figure;
            %scatter(real(X_hat_eq(data_indexes)),
round(imag(X_hat_eq(data_indexes)),3));
```

```
            %axis equal; grid on;
            %title('Equalized Constellation');

                    % --- Plot frequency response off h_n ---
            H_f = fft(h_n, NFFT);  % Frequency-domain representation
            figure
            stem(0:NFFT-1, abs(H_f), 'filled');
            title(sprintf('Channel Frequency Response |H[k]| - BSM %d, OFDM
Symbol %d', 1, i));
            xlabel('Subcarrier Index (k)');
            ylabel('|H[k]|');
            ylim([0 1.1]);
            grid on;
            drawnow;

                    % --- Plot frequency response of h_n ---
            figure
            stem(0:NFFT-1, abs(H_est_prev), 'filled');
            title(sprintf('Estimated Channel Frequency Response |H[k]| - BSM %d,
OFDM Symbol %d', 1, i));
            xlabel('Subcarrier Index (k)');
            ylabel('|H[k]|');
            ylim([0 1.1]);
            grid on;
            drawnow;


            figure;
            plot(angle(H_f), 'b'); hold on;
            plot(angle(H_est_prev), 'r--');
            legend('Angle True', 'Angle Estimated');
            title('Phase of H[k]');

        end
        a=1; %Place breakpoint here to stop for every OFDM Symbol
    end

    % -- Demodulate received symbols for this BSM --
    %demod_bits_all = real(demod_bits_all) > 0;
    %tx_bits_all = X_blocks;
    tx_bits_all = (X_blocks(data_indexes, :) + 1) / 2;
    tx_bits_all = tx_bits_all(:);
    rx_bits_all = demod_bits_all(data_indexes, :);
    rx_bits_all = rx_bits_all(:);

    % DECODE
    %Decoding
    if encoding == "yes"
        %Traceback length (usually 5x(K-1))
        rx_bits_all=vitdec(rx_bits_all, trellis, tbl, 'trunc', 'hard');

    end
    n_errors_all = sum(rx_bits_all ~= BSM_bits');
    BER = n_errors_all/length(BSM_bits);
    %BER = (sum(n_errors_BSM) / nSym_per_BSM)/100;
    BER_vec(bsm_idx) = BER;
    fprintf('BSM %d: Total BER = %.4f\n', bsm_idx, BER);
    % --- Extract decoded BSM coordinates ---
    % Take only the first bits_per_BSM (trim padded/coded bits)
    rx_bits_for_bsm = rx_bits_all(1:bits_per_BSM)';
    [lat_rx, long_rx] = extractLatLongFromBSM(rx_bits_for_bsm);
    fprintf('BSM %d: Decoded Ambulance Position: (%.7f, %.7f)\n', bsm_idx,
lat_rx, long_rx);

    a=1; %Place breakpoint here to stop for every received BSM


end

% ==== PLOT BER ACROSS BSMs ====
figure;
plot(1:nBSMs, BER_vec, '-o', 'LineWidth', 2);
xlabel('BSM Index');
ylabel('BER');
title('BER per BSM (Raytracing Channel, iCDP Equalization)');
grid on;

disp('Simulation complete!');
```

The custom MATLAB script shown above integrates the previously developed transmission and reception system into a complete simulation environment where both the ambulance (transmitter) and the car (receiver) move along predefined trajectories in an urban intersection. The goal is to test the performance of the V2V communication link under realistic propagation conditions, taking into account vehicle motion, multipath effects, and channel estimation.

The script begins by defining simulation parameters, such as carrier frequency, bandwidth, modulation scheme, OFDM settings, pilot and data subcarrier positions, and whether or not forward error correction (FEC) and interleaving are enabled. These parameters ensure the simulated system matches the IEEE 802.11p / DSRC specifications introduced in Section 4.3, while still allowing controlled variations for testing.

Next, the vehicle trajectories are set. The ambulance and the car are assigned start and destination GPS coordinates, and their speeds are specified. Using this information, the script calculates the total travel distance for each vehicle, the corresponding simulation duration, and the number of Basic Safety Messages (BSMs) to be transmitted at the given rate (e.g., 10 Hz). It then generates latitude and longitude points for each OFDM symbol along both vehicle paths, ensuring that the positions are updated at the symbol level for accurate channel modeling.

The propagation environment is modeled using MATLAB's ray tracing functions. A siteviewer object is loaded with an urban map from OpenStreetMap, and a propagationModel is created with configurable parameters such as number of reflections and building material. For each OFDM symbol, the script creates a txsite at the ambulance's current position and an rxsite at the car's position, then computes the multipath channel response via ray tracing. This yields a time-varying, frequency-selective channel that reflects the geometry and materials of the environment.

Within the main simulation loop, each BSM is generated using the ambulance's position at that instant. The message is padded if necessary, optionally encoded with convolutional coding, mapped to BPSK symbols, and organized into OFDM frames with pilot insertion. The IFFT and cyclic prefix addition are applied before passing the signal through the channel model.

On the receiver side, the cyclic prefix is removed, an FFT is applied, and channel equalization is performed using the improved Constructed Data Pilots (iCDP) algorithm described in Section 4.4. This step is crucial for mitigating the effects of the fast-varying vehicular channel. The equalized symbols are then demodulated, optionally decoded, and reassembled into the received BSM bitstream. The script then extracts the decoded latitude and longitude fields to verify that the transmitted position has been recovered correctly.

After processing each BSM, the script computes the bit error rate (BER) for that message and stores the result. At the end of the simulation, it produces a plot of BER versus BSM index, providing a visual performance assessment over the entire vehicle trajectories.

This setup allows testing the complete communication chain—from message generation to application-layer position decoding—under realistic, time-varying urban channel conditions. The flexibility of the script makes it possible to evaluate different modulation schemes, coding rates, equalization strategies, and mobility scenarios, enabling thorough validation of the system's robustness in real-world vehicular environments.

To ensure that the results obtained were not limited to a single use case, the system was tested across a large variety of scenarios. Specifically, almost one hundred different simulations were created by systematically varying key parameters of the environment and the vehicles. These parameters included the geographical location (different urban areas extracted from OpenStreetMap), the number of vehicles present in the scene, their coordinates, and their speeds. In addition, channel-related variables such as the amount of additive noise and the transmission power of the ambulance were also modified between scenarios.

By combining these variations, the testing method covered a wide spectrum of possible real-world situations, ranging from low-traffic urban intersections with minimal interference, to denser traffic environments with higher speeds, stronger multipath effects, and different noise conditions. This extensive scenario library provided a robust validation framework, allowing the system's performance to be assessed under highly diverse operating conditions. As a result, the conclusions drawn from the simulations are not tied to a single artificial configuration, but instead represent the expected behavior of the system in a broad range of realistic urban settings.

5.3.2 TESTING RESULTS

After running the previous script with some given coordinates for the vehicles, the generated initial scenario is displayed in Figure 15.

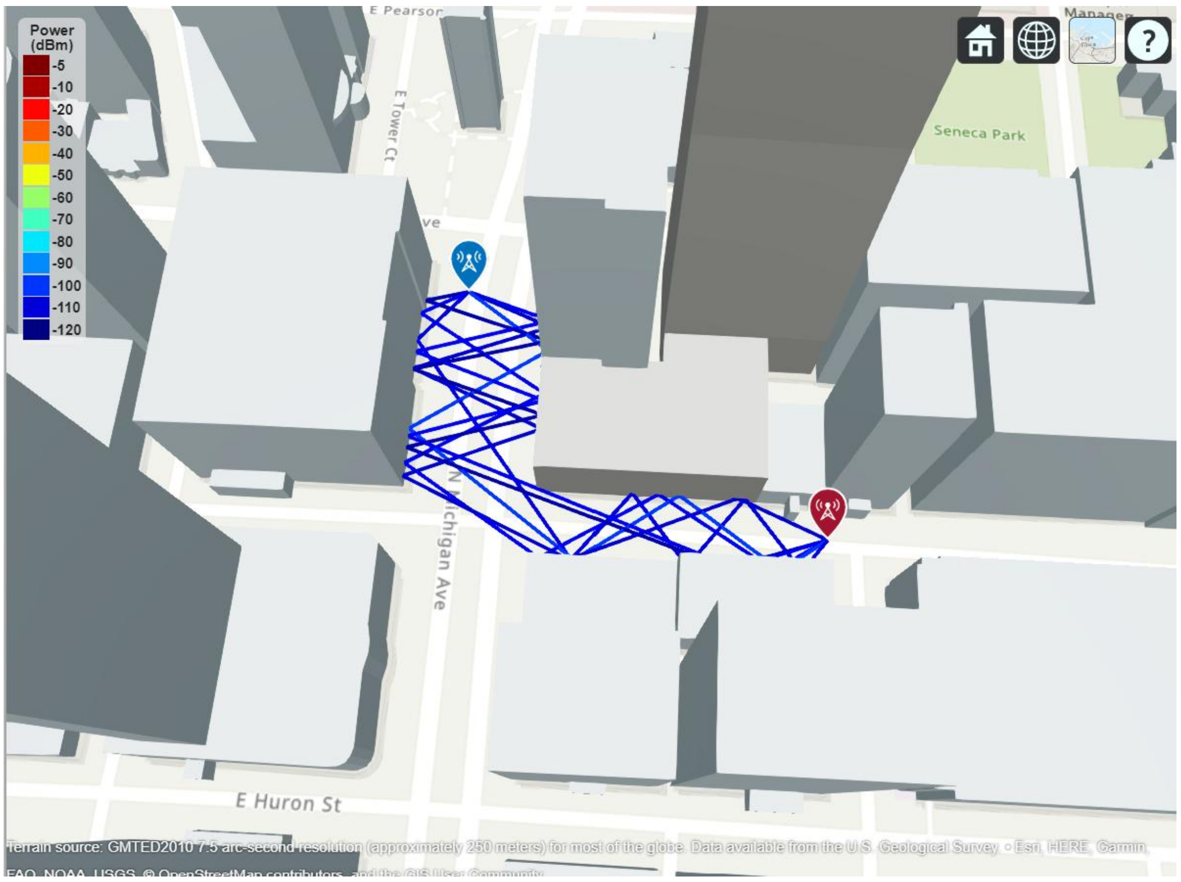

Power (dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
E Pearson
E Tower Ct
Seneca Park
N Michigan Ave
E Huron St
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Esri, HERE, Garmin, FAO, NOAA, USGS, © OpenStreetMap contributors, and the GIS User Community

Figure 15. Generated Initial scenario

This corresponds to a real intersection in the center of Chicago, where an ambulance (in red) is heading West and a car (in blue) is heading South. The speeds of both the ambulance and the car are 20 m/s, or 45 mph, or 72kph. The simulation will run in steps of 8us (duration of one OFDM symbol), calculating the new position of the vehicles at each step, and sending the corresponding OFDM symbol through the new simulated channel.

For each OFDM symbol, the script displays the comparison between the true channel distortion, and the estimated channel distortion. This serves as a perfect way of determining how well the system works when it comes to compensating for the effects caused by the transmission channel. The script also displays the BER (Bit Error Rate) for each OFDM symbol.

For the first OFDM symbol, Figure 16 shows the obtained

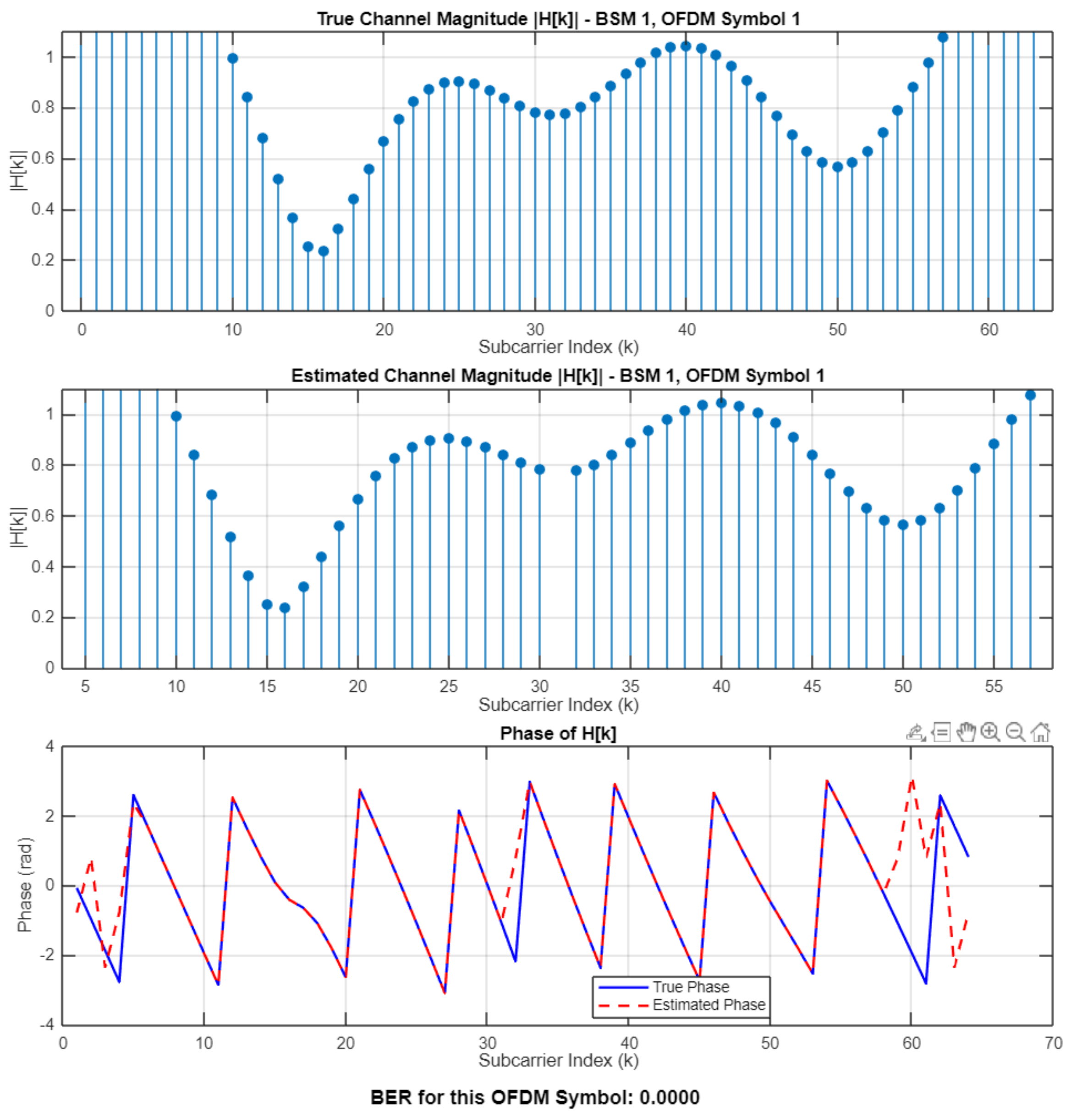


Figure 16. First symbol results

This shows a perfect match between the true channel response and the one estimated by the system. In consequence, the BER for this OFDM symbol is null, so every single bit is correctly recovered at the receiver. However, it is important to keep in mind that this is totally normal, because the first symbol is a pilot, so the receiver is able to make a perfect channel estimation. This first estimation serves as a base for the CDP algorithm to recover the following symbols, which contain real unknown data. Thus, the results for the following symbols show the real performance of the system.

On the second OFDM symbol, this result shown in Figure 17 is observed:

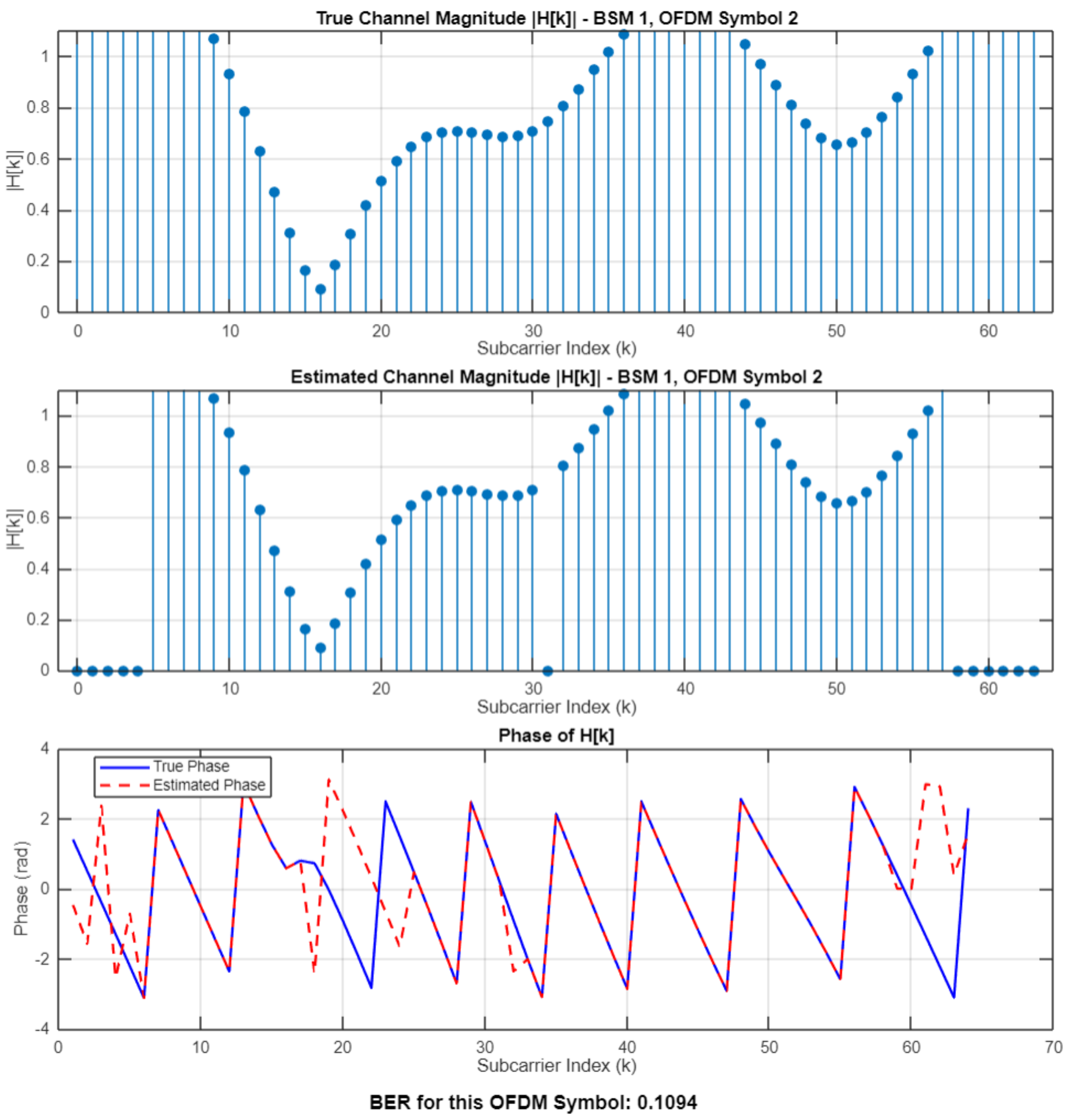


Figure 17. Second symbol results

The vehicles have now moved, so the signal propagation has slightly changed, causing the channel response to vary. Although the system is able to estimate the channel almost perfectly, the slight error in the phase calculation causes a 10% bit error rate. This is caused by a change in the channel that is too sudden for the CDP algorithm to keep up. Nevertheless, the BER is still acceptable, as it is always important to note that a BSM is composed of 6 or 7 symbols, so a 10% error in one of them is not so relevant.

For the third symbol, the results shown in Figure 18 are observed:

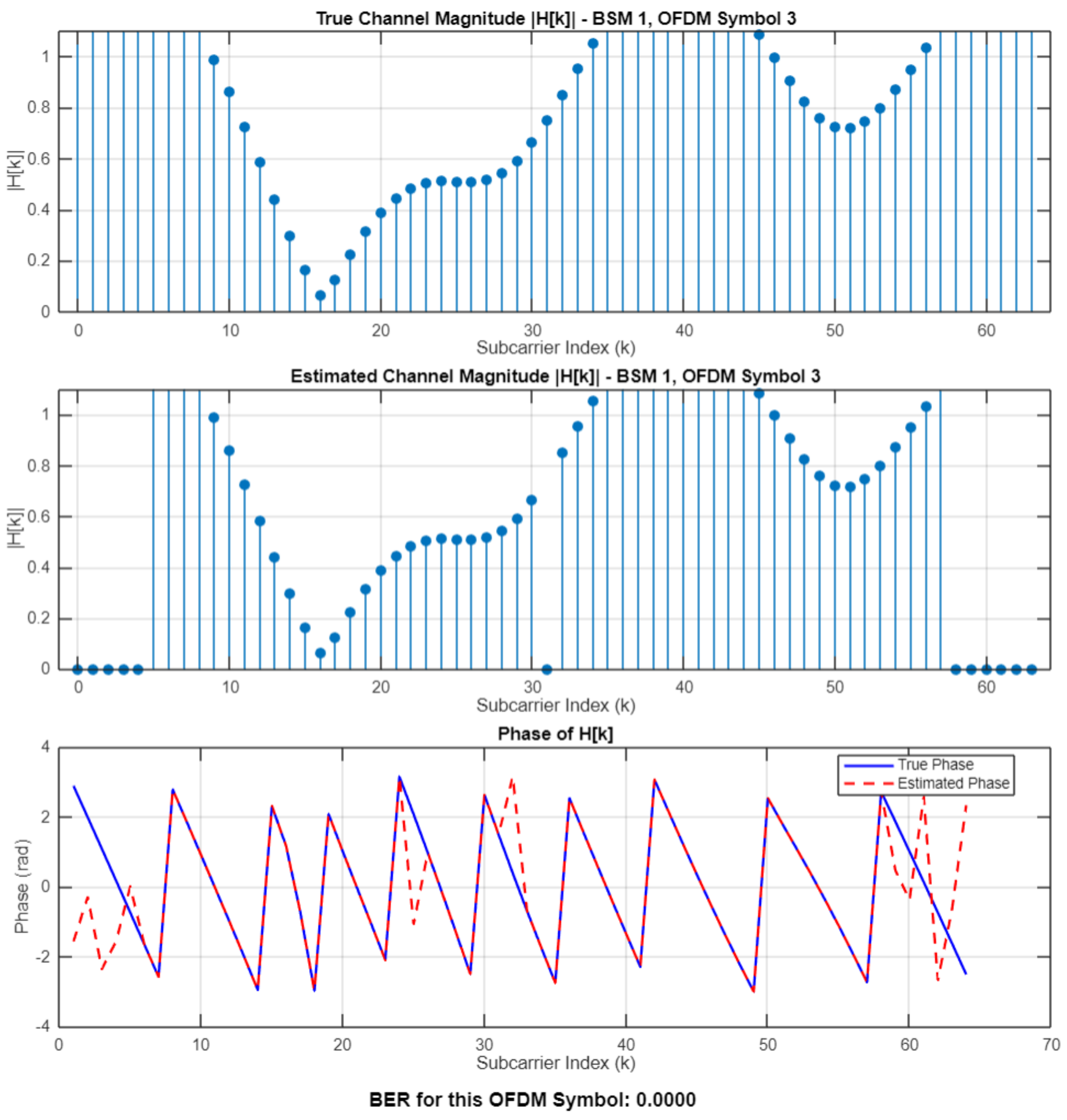
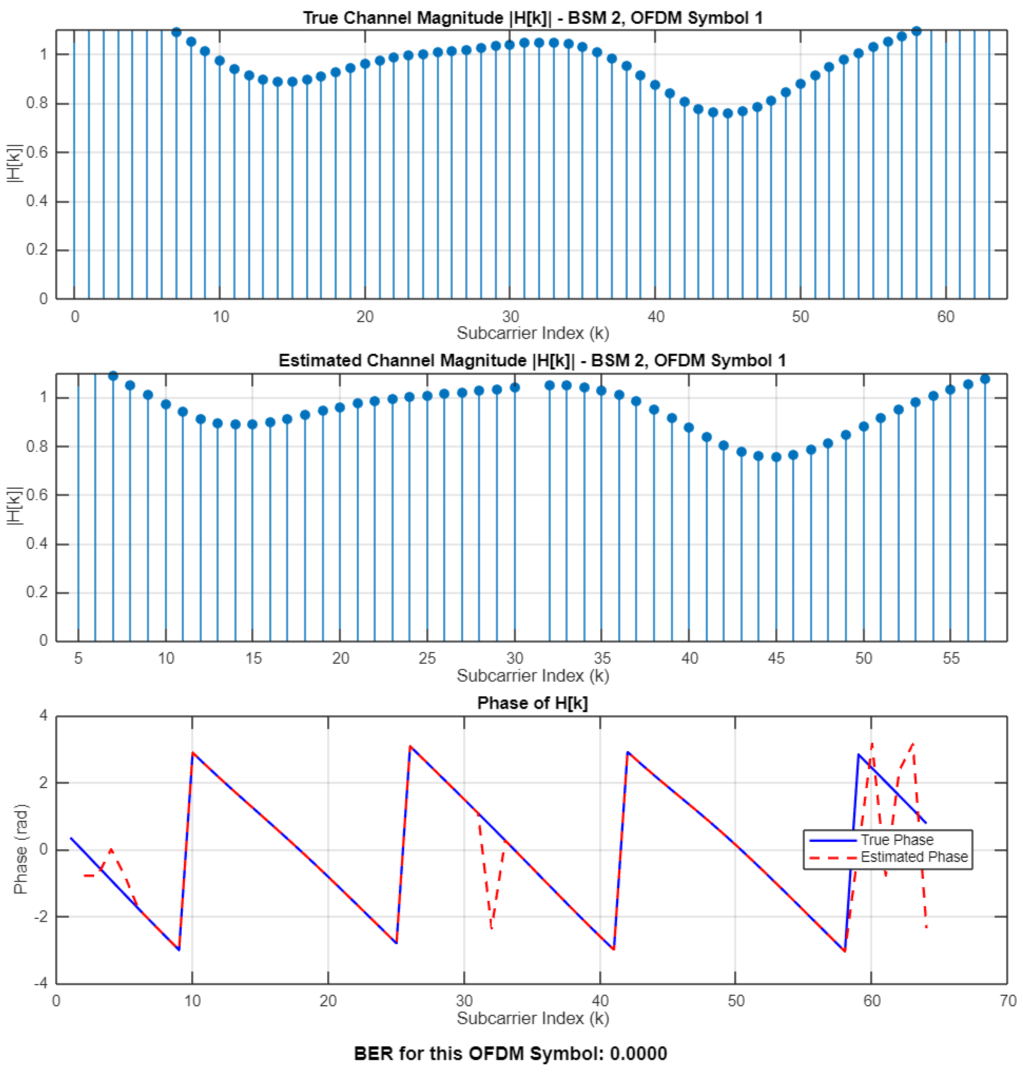
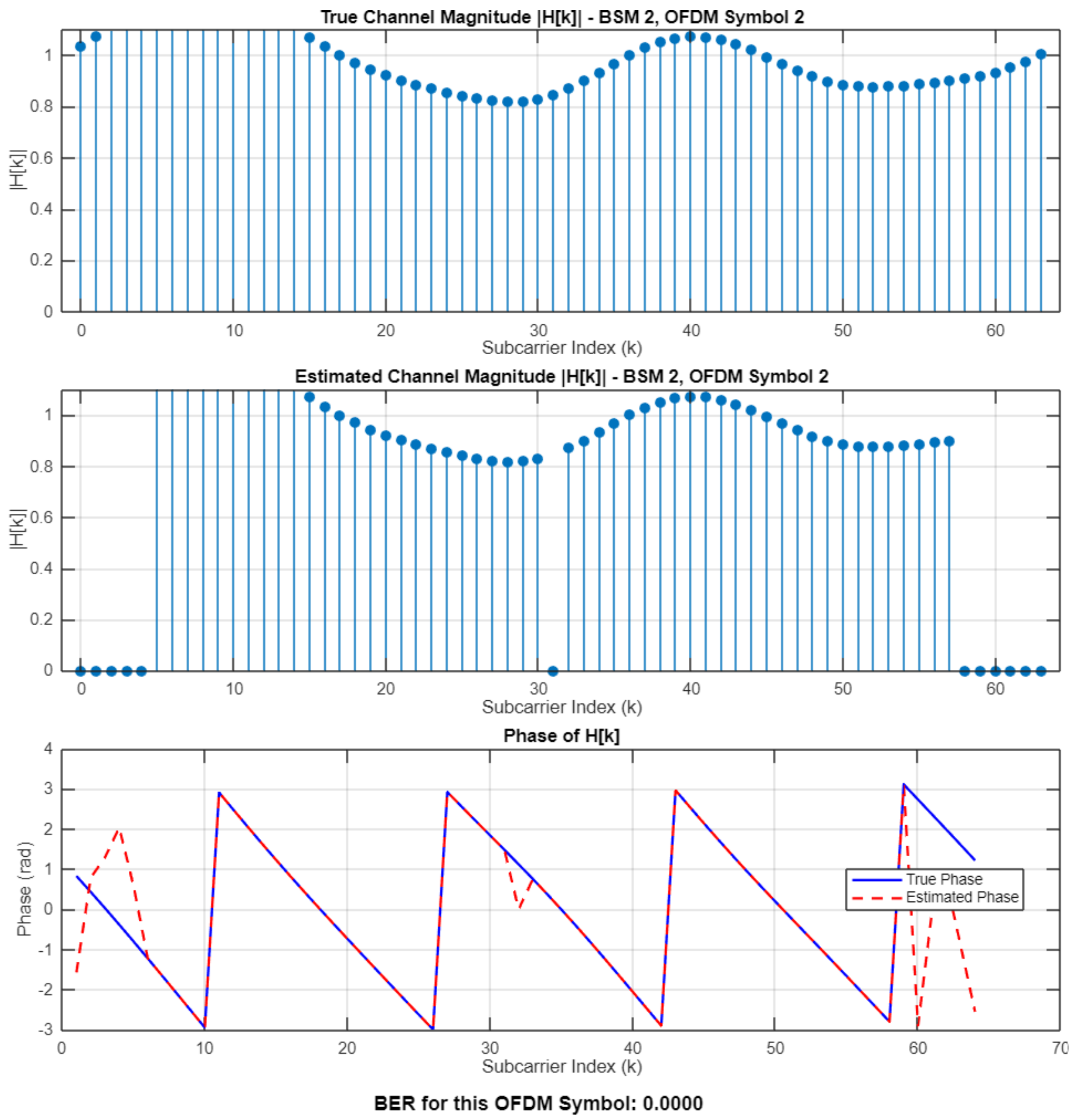


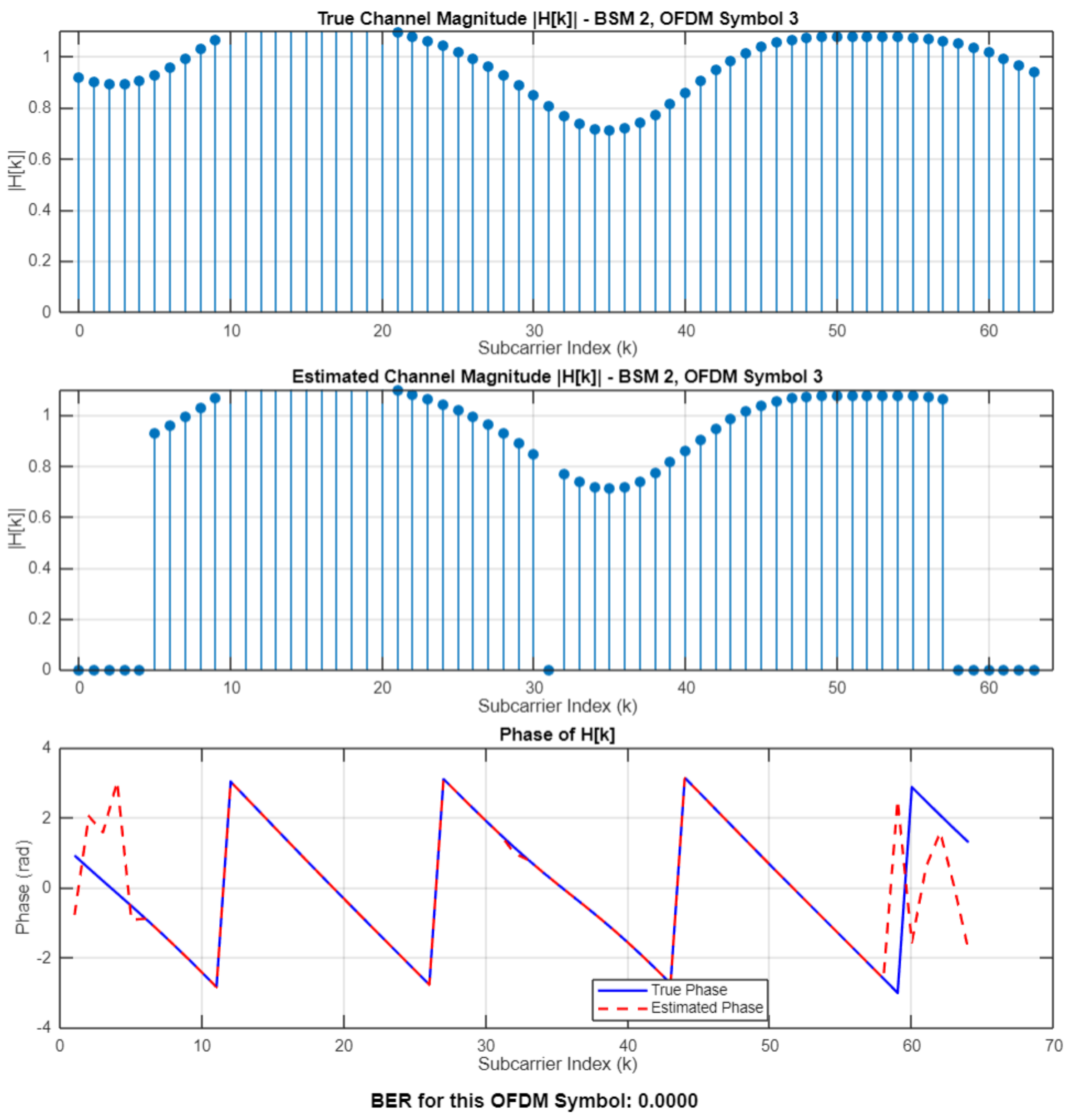
Figure 18. Third symbol results

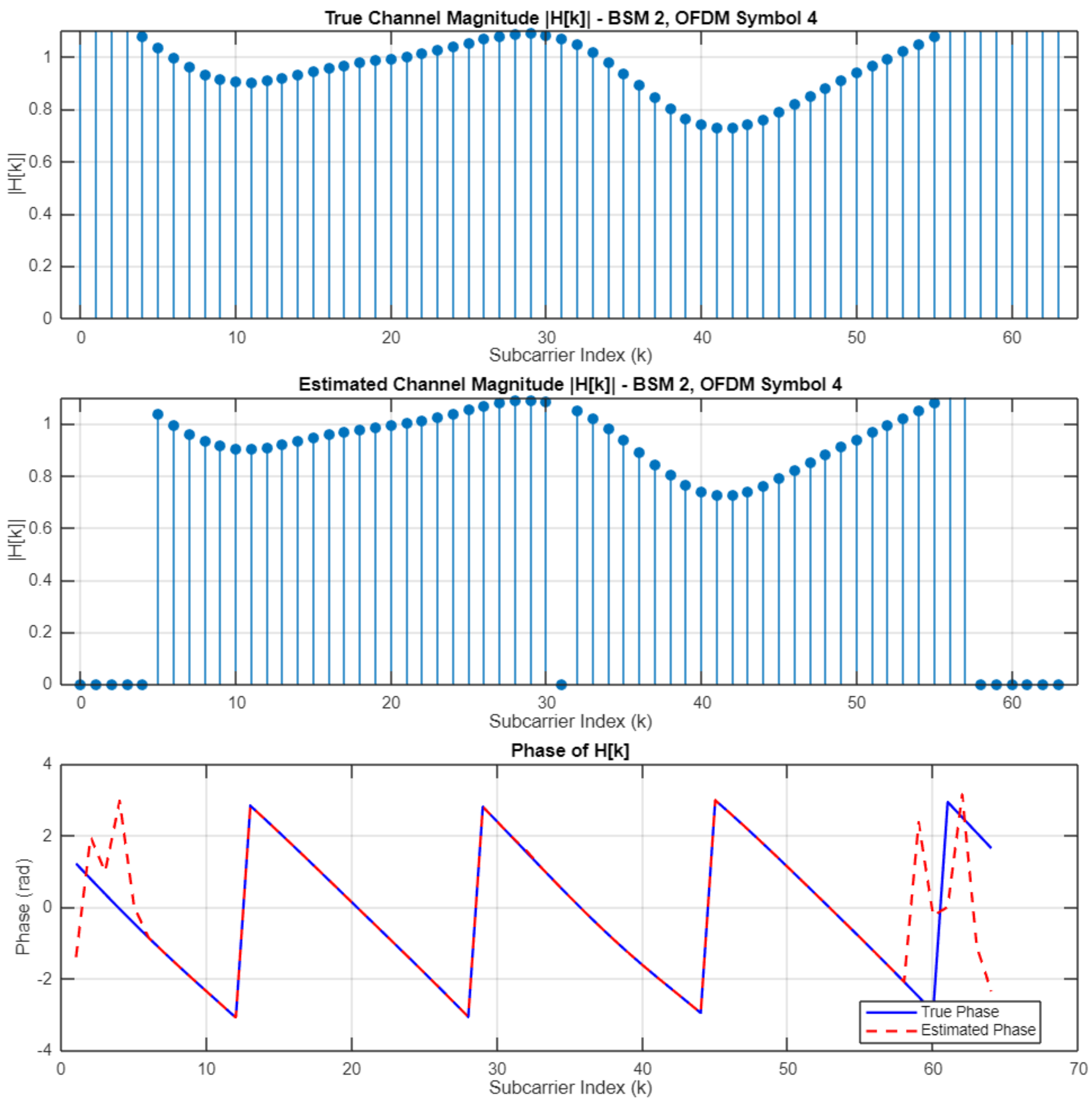
Now, the channel has also changed due to the vehicle's movement. However, this time, thanks to the CDP algorithm that adapted the channel estimation with the second symbol, the result is that the third symbol is recovered perfectly, with a 0% BER. This is a perfect example of the CDP algorithm's capabilities when working with scenarios that show a fast channel variation.

If the simulation is left running, this script allows tracking the performance of the system at every moment. For example, skipping to the 7th transmitted symbol, this now corresponds to the second BSM. Now, the ambulance starts transmitting the updated coordinates. The total output for the 6 symbols of the second BSM is shown in Figure 19:

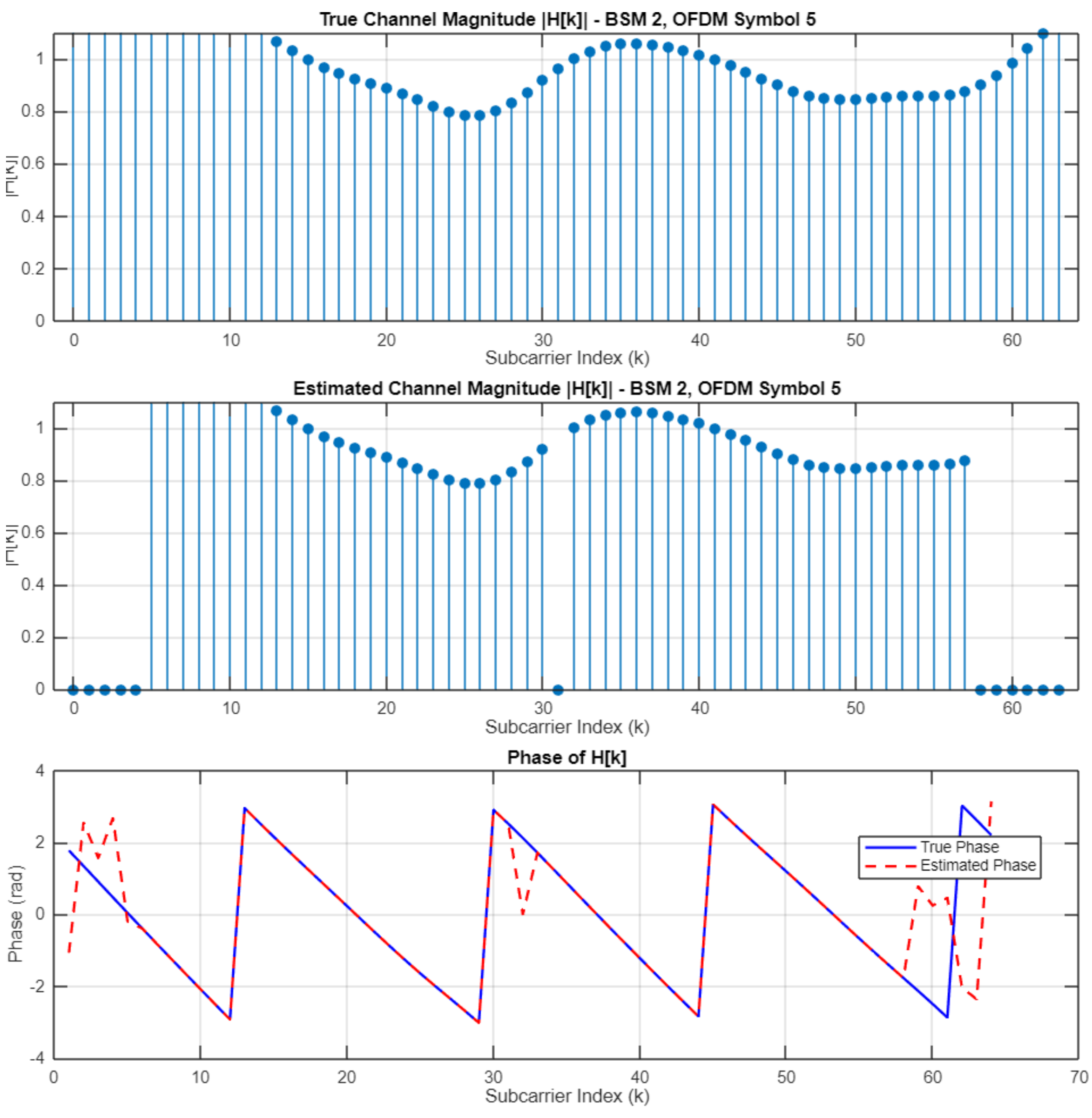



True Channel Magnitude |H[k]| - BSM 2, OFDM Symbol 2
|H[k]|
Subcarrier Index (k)
Estimated Channel Magnitude |H[k]| - BSM 2, OFDM Symbol 2
|H[k]|
Subcarrier Index (k)
Phase of H[k]
Phase (rad)
Subcarrier Index (k)
True Phase
Estimated Phase
BER for this OFDM Symbol: 0.0000


True Channel Magnitude |H[k]| - BSM 2, OFDM Symbol 3
|H[k]|
Subcarrier Index (k)
Estimated Channel Magnitude |H[k]| - BSM 2, OFDM Symbol 3
|H[k]|
Subcarrier Index (k)
Phase of H[k]
Phase (rad)
Subcarrier Index (k)
True Phase
Estimated Phase
BER for this OFDM Symbol: 0.0000



BER for this OFDM Symbol: 0.0000



BER for this OFDM Symbol: 0.0000

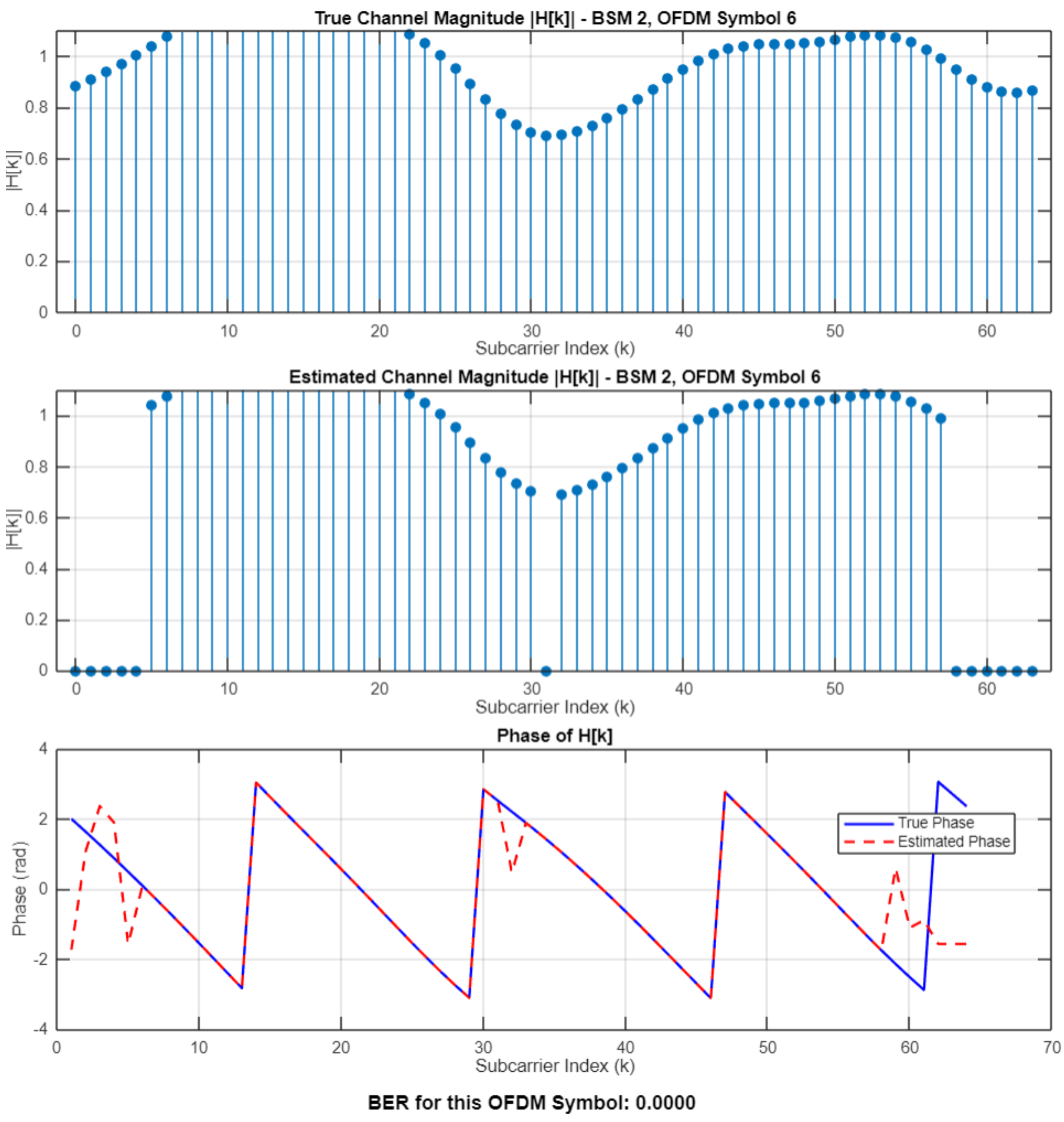


Figure 19. Result for second BSM (all symbols)

As can be seen in the figures above, all the OFDM symbols are recovered perfectly (0% BER), despite the channel variations. This is one of the many examples that confirms that the designed system is working as expected.

5.3.3 SUCCESS PHILOSOPHY

The primary objective of the proposed system is not to achieve perfect recovery of every transmitted Basic Safety Message (BSM), but rather to reliably convey the ambulance's position often enough for the receiver to make an informed decision. In the context of this application, even a small proportion of correctly decoded BSMs can be sufficient for success. This is because the essential information (the latitude and longitude of the ambulance) is contained in each BSM, and a handful of correct coordinates is enough to reconstruct the ambulance's trajectory with high confidence.

When a BSM is corrupted due to channel impairments, equalization errors, or noise, the resulting decoded coordinates are typically unrealistic: they may indicate a physically impossible location, or a position that is far outside the expected vicinity of the intersection. Such anomalies can be easily filtered out by the receiver, either by applying basic sanity checks (e.g., rejecting coordinates beyond a certain distance threshold) or by analyzing the consistency of the reported trajectory over time.

Consequently, the success criterion for this system is not the percentage of BSMs recovered without error, but rather the presence of enough valid BSMs to reconstruct a plausible and continuous path of the ambulance's movement. For example, if only 5 BSMs are correctly received out of 100 transmissions, those 5 points may still form a coherent, straight-line trajectory that accurately indicates the ambulance's approach direction and speed.

Given that BSMs are transmitted every 10 ms, the system benefits from a high temporal redundancy. In real-world scenarios, this means that even under severe channel degradation, the probability of receiving several error-free BSMs within a short time window is very high. As a result, the system can reliably achieve its main purpose (alerting nearby vehicles to the ambulance's presence and direction) without requiring continuous, flawless communication.

5.4 FINAL IMPLEMENTATION

The program described in Section 5.3 has proven reliable and effective. However, it is not the visual and interactive application that this project is aiming for. In the end, the purpose of this project is to design a program that lets the user place an ambulance and some vehicles inside a simulated urban scenario, with their respective speeds and movement directions, and show the real-time results. The final application must display the program's output, which is what the drivers would see in their car screens, showing the direction where the ambulance is coming from, and how far away it is at that moment.

For this last part, the MATLAB App Designer has been used. Due to the computational complexity of the simulation, the final product has been divided into 2 different applications. The first app is used to create scenarios and run them, storing all the results. The second app displays the results in an interactive and friendly way.

5.4.1 SCENARIO CREATION APP

The Scenario Creation App is the first stage of the final implementation. Its purpose is to allow the user to design and configure a complete test scenario through a graphical interface. Instead of manually editing parameters in a script, the user can now visually place the ambulance and any number of cars inside a realistic urban map, set their starting and destination points, and define their movement speeds.

The application loads a geo-referenced map of the chosen city area, onto which the user can click to select coordinates. Each vehicle's origin and destination are marked with different symbols and connected by a dashed line, visually indicating the planned trajectory. The user can specify the number of cars in the scenario and switch between selecting the ambulance or one of the cars from a drop-down menu.

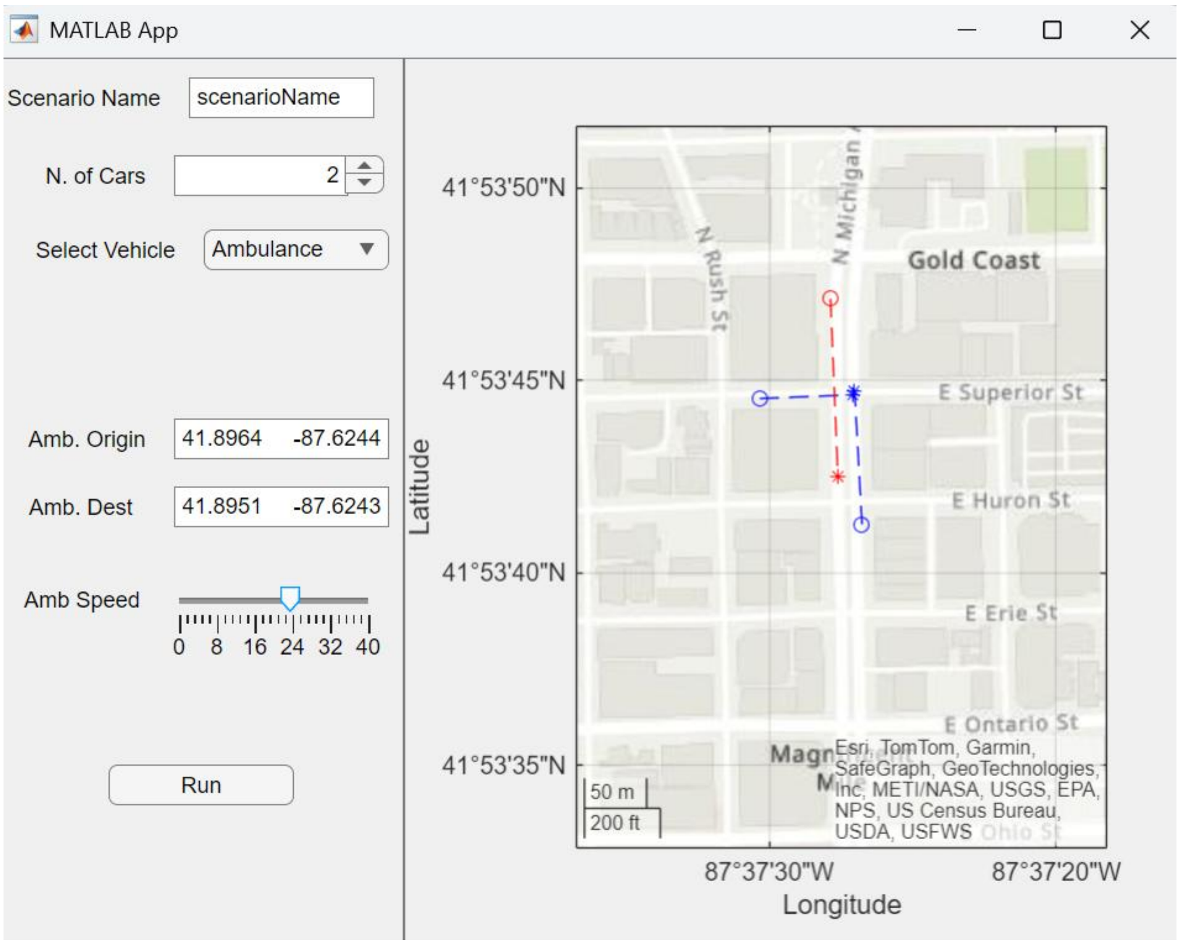
Once the scene is configured, the Run button launches the simulation for every vehicle. For each car, the app runs the full communication process with the moving ambulance, using the underlying simulation engine (described in Section 5.3). It stores the results (including the received ambulance coordinates, Bit Error Rates (BERs), channel data, and SNR values) into separate `.mat` files. These results can later be loaded into the second application for visualization and analysis.

This app focuses purely on scenario creation and execution**,** without overwhelming the user with technical details of the underlying simulation. Its simple interface hides the complexity of the signal processing, making it possible to create complex, multi-vehicle communication experiments with only a few clicks.

The code that makes all this possible is shown here:

```
classdef app1 < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                      matlab.ui.Figure
        GridLayout                    matlab.ui.container.GridLayout
        LeftPanel                     matlab.ui.container.Panel
        ScenarioNameEditField         matlab.ui.control.EditField
        ScenarioNameEditFieldLabel    matlab.ui.control.Label
        NofCarsSpinner                matlab.ui.control.Spinner
        NofCarsSpinnerLabel           matlab.ui.control.Label
        SelectVehicleDropDown         matlab.ui.control.DropDown
        SelectVehicleDropDownLabel    matlab.ui.control.Label
        AmbDestEditField              matlab.ui.control.EditField
        AmbDestEditFieldLabel         matlab.ui.control.Label
        AmbOriginEditField            matlab.ui.control.EditField
        AmbOriginEditFieldLabel       matlab.ui.control.Label
        RunButton                     matlab.ui.control.Button
        AmbSpeedSlider                matlab.ui.control.Slider
        AmbSpeedSliderLabel           matlab.ui.control.Label
        RightPanel                    matlab.ui.container.Panel
    end

    % Properties that correspond to apps with auto-reflow
    properties (Access = private)
        onePanelWidth = 576;
    end


    properties (Access = public)
        f = 5.9e9;
        tx_power = 1;                % Watts
        B = 10e6;                    % Bandwidth Hz
        map = 'maps/map_chicago.osm';
        lat_limits = [41.893410 41.896720];
        lon_limits = [-87.626877 -87.621727];
        encoding = "no";
        interleaving = "no";
        view_plots = "no";
        view_scene = "no";
        ch_est_method = 'iCDP';     % Per-symbol equalization
        simulate_channel_distortion = "yes";
        mod_order = 1;              % BPSK
        ambulance_speed = 20;       % m/s
        car_speed = 10;             % m/s
        BSMs_per_second = 100;       % BSMs every 100 ms
        max_number_reflections = 3;
        n_cars = 1; % Number of cars to place in the scenario
        selected_vehicle = 0;

        % OFDM parameters
        nBitsSymbBPSK = 48;
        NFFT = 64;
        nSubCar = 52;
        Ncp = 16;
        Tofdm = 8e-6;    % 8 us per OFDM symbol
        n_taps = 8;      % Required by equalization fn
        pilot_indexes = [11, 25, 39, 53];
        data_indexes = [6:10, 12:24, 26:31, 33:38, 40:52, 54:58];
        symbol_book = [-1; 1];      % BPSK
        R_c = 1;
        K = 7;
        tbl = 32;
        ambulance_orig = [41.895613,-87.624195];
        ambulance_dest = [41.895125,-87.624174];
        %car_orig = [41.895900,-87.624206];
        %car_dest = [41.895434,-87.624190];
        bits_per_BSM = 266;
        gx
        cars_orig = [];
        cars_dest = [];
        BER_cars = [];
        amb_coords_rx = [];

        scenario_name;

    end


    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
            app.gx = geoaxes(app.RightPanel);
            set(0, 'CurrentFigure', app.UIFigure)       % Make fig current
            %gx = geoaxes(app.RightPanel);
            %gx.HandleVisibility
            hold(app.gx, "on")
            geobasemap(app.gx, "topographic")
            lat = [40 -74]; % Example latitudes
            lon = [38 -72]; % Example longitudes
            geoplot(app.gx, lat, lon, 'o');
            geolimits(app.gx, app.lat_limits, app.lon_limits);

            %app.SelectVehicleDropDown.Items = {'Ambulance', 'Car1'};
        end

        % Callback function
        function StartAmbLonEditFieldValueChanged(app, event)
            value = app.StartAmbLonEditField.Value;
            app.ambulance_orig(2) = value;

        end

        % Callback function
        function StartAmbLatEditFieldValueChanged(app, event)
            value = app.StartAmbLatEditField.Value;
            app.ambulance_orig(1) = value;
        end

        % Value changed function: AmbSpeedSlider
        function AmbSpeedSliderValueChanged(app, event)
            value = app.AmbSpeedSlider.Value;
            app.ambulance_speed = value;
        end

        % Button pushed function: RunButton
        function RunButtonPushed(app, event)

            for car_index=1:app.n_cars
                fprintf('Running Simulation for Car %d', car_index);
                [BER_vec, ambulance_lats_rx, ambulance_longs_rx,
ambulance_lats_tx, ambulance_longs_tx, car_lats, car_longs, h_ns, SNR_dBs, nBSMs,
nSym_per_BSM] = main_9(app, app.cars_orig(car_index, :), app.cars_dest(car_index,
:));
                fprintf('Finished Simulation for Car %d', car_index);
                BER_vec;
                %plot(app.UIAxes, BER_vec)
                speed = app.ambulance_speed;

                filename =
"results/"+app.scenario_name+"_"+num2str(car_index)+".mat";
                %save(filename, "BER_vec", "ambulance_lats_rx",
"ambulance_longs_rx", "ambulance_lats_tx", "ambulance_longs_tx", "car_orig",
"car_dest", "")
                %save(filename);
                save_scenario(filename, app, BER_vec, ambulance_lats_rx,
ambulance_longs_rx, ambulance_lats_tx, ambulance_longs_tx, car_lats, car_longs,
h_ns, SNR_dBs, nBSMs, nSym_per_BSM, car_index);
            end
        end

        % Callback function
        function NofCarsEditFieldValueChanged(app, event)

        end

        % Value changed function: SelectVehicleDropDown
        function SelectVehicleDropDownValueChanged(app, event)
            value = app.SelectVehicleDropDown.Value;
            app.selected_vehicle = value;

            % Set up figure handle visibility, run ginput, and return state
            fhv = app.UIFigure.HandleVisibility;        % Current status
            app.UIFigure.HandleVisibility = 'callback'; % Temp change (or, 'on')
            set(0, 'CurrentFigure', app.UIFigure)       % Make fig current


            % Use ginput to select points on the plot
            [x1 y1] = ginput(1);

            if app.selected_vehicle==0
                geoplot(app.gx, x1, y1, 'ro');
                app.ambulance_orig = [x1 y1];
                app.AmbOriginEditField.Value = num2str(app.ambulance_orig);
            else
                geoplot(app.gx, x1, y1, 'bo')
                app.cars_orig(app.selected_vehicle, :) = [x1 y1];
            end

            [x2 y2] = ginput(1);
            if app.selected_vehicle == 0
                geoplot(app.gx, x2, y2, 'r*');
                app.ambulance_dest = [x2 y2];
                app.AmbDestEditField.Value = num2str(app.ambulance_dest);
            else
                geoplot(app.gx, x2, y2, 'b*')
                app.cars_dest(app.selected_vehicle, :) = [x2 y2];
            end

            if app.selected_vehicle == 0
               %geoshow([x1 x2], [y1 y2], 'DisplayType', 'line', 'Marker', '>',
'Color', 'r')
               geoplot(app.gx, [x1 x2], [y1 y2], 'r--')
            else
               %geoshow([x1 x2], [y1 y2], 'DisplayType', 'line', 'Marker', '>',
'Color', 'b')
               geoplot(app.gx, [x1 x2], [y1 y2], 'b--')

            end
            % Display the coordinates of the selected points
            fprintf('Selected points: (%f, %f)\n', [x1 y1 x2 y2]);
            %[x,y,b] = ginput(1);                        % run ginput

            app.UIFigure.HandleVisibility = fhv;        % return original state







        end

        % Value changed function: NofCarsSpinner
        function NofCarsSpinnerValueChanged(app, event)
            value = app.NofCarsSpinner.Value;
            app.SelectVehicleDropDown.Items = [{'Ambulance'}];
            app.SelectVehicleDropDown.ItemsData = [0];
            app.n_cars = value;
            %app.selected_vehicle = 0;
            for i=2:app.n_cars+1
                s = 'Car0'
                s(length(s)) = num2str(i-1);
                option = {s};
                app.SelectVehicleDropDown.Items
                app.SelectVehicleDropDown.Items =
[app.SelectVehicleDropDown.Items, option];
                app.SelectVehicleDropDown.ItemsData(i) = i-1;
            end
        end

        % Value changed function: ScenarioNameEditField
        function ScenarioNameEditFieldValueChanged(app, event)
            value = app.ScenarioNameEditField.Value;
            app.scenario_name = value;
        end

        % Changes arrangement of the app based on UIFigure width
        function updateAppLayout(app, event)
            currentFigureWidth = app.UIFigure.Position(3);
            if(currentFigureWidth <= app.onePanelWidth)
                % Change to a 2x1 grid
                app.GridLayout.RowHeight = {480, 480};
                app.GridLayout.ColumnWidth = {'1x'};
                app.RightPanel.Layout.Row = 2;
                app.RightPanel.Layout.Column = 1;
            else
                % Change to a 1x2 grid
                app.GridLayout.RowHeight = {'1x'};
                app.GridLayout.ColumnWidth = {220, '1x'};
                app.RightPanel.Layout.Row = 1;
                app.RightPanel.Layout.Column = 2;
            end
        end
    end
    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.AutoResizeChildren = 'off';
            app.UIFigure.Position = [100 100 640 480];
            app.UIFigure.Name = 'MATLAB App';
            app.UIFigure.SizeChangedFcn = createCallbackFcn(app,
@updateAppLayout, true);

            % Create GridLayout
            app.GridLayout = uigridlayout(app.UIFigure);
            app.GridLayout.ColumnWidth = {220, '1x'};
            app.GridLayout.RowHeight = {'1x'};
            app.GridLayout.ColumnSpacing = 0;
            app.GridLayout.RowSpacing = 0;
            app.GridLayout.Padding = [0 0 0 0];
            app.GridLayout.Scrollable = 'on';

            % Create LeftPanel
            app.LeftPanel = uipanel(app.GridLayout);
            app.LeftPanel.Layout.Row = 1;
            app.LeftPanel.Layout.Column = 1;

            % Create AmbSpeedSliderLabel
            app.AmbSpeedSliderLabel = uilabel(app.LeftPanel);
            app.AmbSpeedSliderLabel.HorizontalAlignment = 'right';
            app.AmbSpeedSliderLabel.Position = [10 177 68 22];
            app.AmbSpeedSliderLabel.Text = 'Amb Speed';

            % Create AmbSpeedSlider
            app.AmbSpeedSlider = uislider(app.LeftPanel);
            app.AmbSpeedSlider.Limits = [0 40];
            app.AmbSpeedSlider.ValueChangedFcn = createCallbackFcn(app,
@AmbSpeedSliderValueChanged, true);
            app.AmbSpeedSlider.Position = [99 186 102 3];
            app.AmbSpeedSlider.Value = 30;

            % Create RunButton
            app.RunButton = uibutton(app.LeftPanel, 'push');
            app.RunButton.ButtonPushedFcn = createCallbackFcn(app,
@RunButtonPushed, true);
            app.RunButton.Position = [61 77 100 22];
            app.RunButton.Text = 'Run';

            % Create AmbOriginEditFieldLabel
            app.AmbOriginEditFieldLabel = uilabel(app.LeftPanel);
            app.AmbOriginEditFieldLabel.HorizontalAlignment = 'right';
            app.AmbOriginEditFieldLabel.Position = [13 264 68 22];
            app.AmbOriginEditFieldLabel.Text = 'Amb. Origin';

            % Create AmbOriginEditField
            app.AmbOriginEditField = uieditfield(app.LeftPanel, 'text');
            app.AmbOriginEditField.Position = [96 264 116 22];

            % Create AmbDestEditFieldLabel
            app.AmbDestEditFieldLabel = uilabel(app.LeftPanel);
            app.AmbDestEditFieldLabel.HorizontalAlignment = 'right';
            app.AmbDestEditFieldLabel.Position = [13 227 61 22];
            app.AmbDestEditFieldLabel.Text = 'Amb. Dest';

            % Create AmbDestEditField
            app.AmbDestEditField = uieditfield(app.LeftPanel, 'text');
            app.AmbDestEditField.Position = [96 227 116 22];

            % Create SelectVehicleDropDownLabel
            app.SelectVehicleDropDownLabel = uilabel(app.LeftPanel);
            app.SelectVehicleDropDownLabel.HorizontalAlignment = 'right';
            app.SelectVehicleDropDownLabel.Position = [17 366 80 22];
            app.SelectVehicleDropDownLabel.Text = 'Select Vehicle';

            % Create SelectVehicleDropDown
            app.SelectVehicleDropDown = uidropdown(app.LeftPanel);
            app.SelectVehicleDropDown.Items = {'Ambulance', 'Car1'};
            app.SelectVehicleDropDown.ItemsData = [0 1];
            app.SelectVehicleDropDown.ValueChangedFcn = createCallbackFcn(app,
@SelectVehicleDropDownValueChanged, true);
            app.SelectVehicleDropDown.Position = [112 366 100 22];
            app.SelectVehicleDropDown.Value = 0;

            % Create NofCarsSpinnerLabel
            app.NofCarsSpinnerLabel = uilabel(app.LeftPanel);
            app.NofCarsSpinnerLabel.HorizontalAlignment = 'right';
            app.NofCarsSpinnerLabel.Position = [22 406 59 22];
            app.NofCarsSpinnerLabel.Text = 'N. of Cars';

            % Create NofCarsSpinner
            app.NofCarsSpinner = uispinner(app.LeftPanel);
            app.NofCarsSpinner.ValueChangedFcn = createCallbackFcn(app,
@NofCarsSpinnerValueChanged, true);
            app.NofCarsSpinner.Position = [96 406 116 22];
            app.NofCarsSpinner.Value = 1;

            % Create ScenarioNameEditFieldLabel
            app.ScenarioNameEditFieldLabel = uilabel(app.LeftPanel);
            app.ScenarioNameEditFieldLabel.HorizontalAlignment = 'right';
            app.ScenarioNameEditFieldLabel.Position = [1 448 88 22];
            app.ScenarioNameEditFieldLabel.Text = 'Scenario Name';

            % Create ScenarioNameEditField
            app.ScenarioNameEditField = uieditfield(app.LeftPanel, 'text');
            app.ScenarioNameEditField.ValueChangedFcn = createCallbackFcn(app,
@ScenarioNameEditFieldValueChanged, true);
            app.ScenarioNameEditField.Position = [104 448 100 22];

            % Create RightPanel
            app.RightPanel = uipanel(app.GridLayout);
            app.RightPanel.Layout.Row = 1;
            app.RightPanel.Layout.Column = 2;

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = app1

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

By running this code, the app that is generated looks like this:


MATLAB App

Scenario Name: scenarioName
N. of Cars: 2
Select Vehicle: Ambulance
Amb. Origin: 41.8964 -87.6244
Amb. Dest: 41.8951 -87.6243
Amb Speed

Figure 20. Scenario creation App

In the example shown in Figure 20, a scenario has been configured with an ambulance traveling South, and 2 cars (one traveling East and the other one traveling North). By clicking on the Run button, the simulation begins and, after a few minutes, the results are stored into multiple .mat files (one for each car). Then, the results can be visualized using the second App.

5.4.2 RESULTS VIEWER APP

The Results Viewer App is the second stage of the final implementation. Its main function is to load and display the scenarios previously generated with the Scenario Creation App, offering an interactive and visual representation of the simulation results. It provides the same perspective that drivers would have in a real vehicle, showing them the ambulance's relative position and distance over time.

Once the user selects and loads the stored .mat files containing simulation results, the app displays a map view of the urban environment (Figure 21). The ambulance (red) and the cars (blue) are shown with their origins, destinations, and current positions, and their movement is animated through the time slider at the top. The map also includes dashed lines representing each vehicle's trajectory so far.

On the left panel, the app lists all loaded scenarios and provides gauges showing the real-time speeds of both the ambulance and the currently selected car, as well as the Signal to Noise Ratio (SNR) of the received signal for the selected car at the selected moment. The "Show Scenario" button triggers a 3D propagation view with ray tracing, allowing inspection of the communication channel in that exact moment.

At the bottom, two synchronized visualizations give deeper insights:

- Ambulance Detection Dashboard (left): Shows the ambulance's detected direction relative to the car's heading, with the distance updated in meters.

- BER per BSM Plot (right): Displays the Bit Error Rate for each received Basic Safety Message. Messages with BER = 0 (green) are perfectly decoded, while red bars indicate messages with some level of error. This visualization allows the user to immediately see the quality of the communication over time.

Each car has its own dedicated tab, so the user can switch between vehicles and see the results from different perspectives. The playback can be paused, reversed, or advanced manually to inspect specific moments of the simulation in detail.

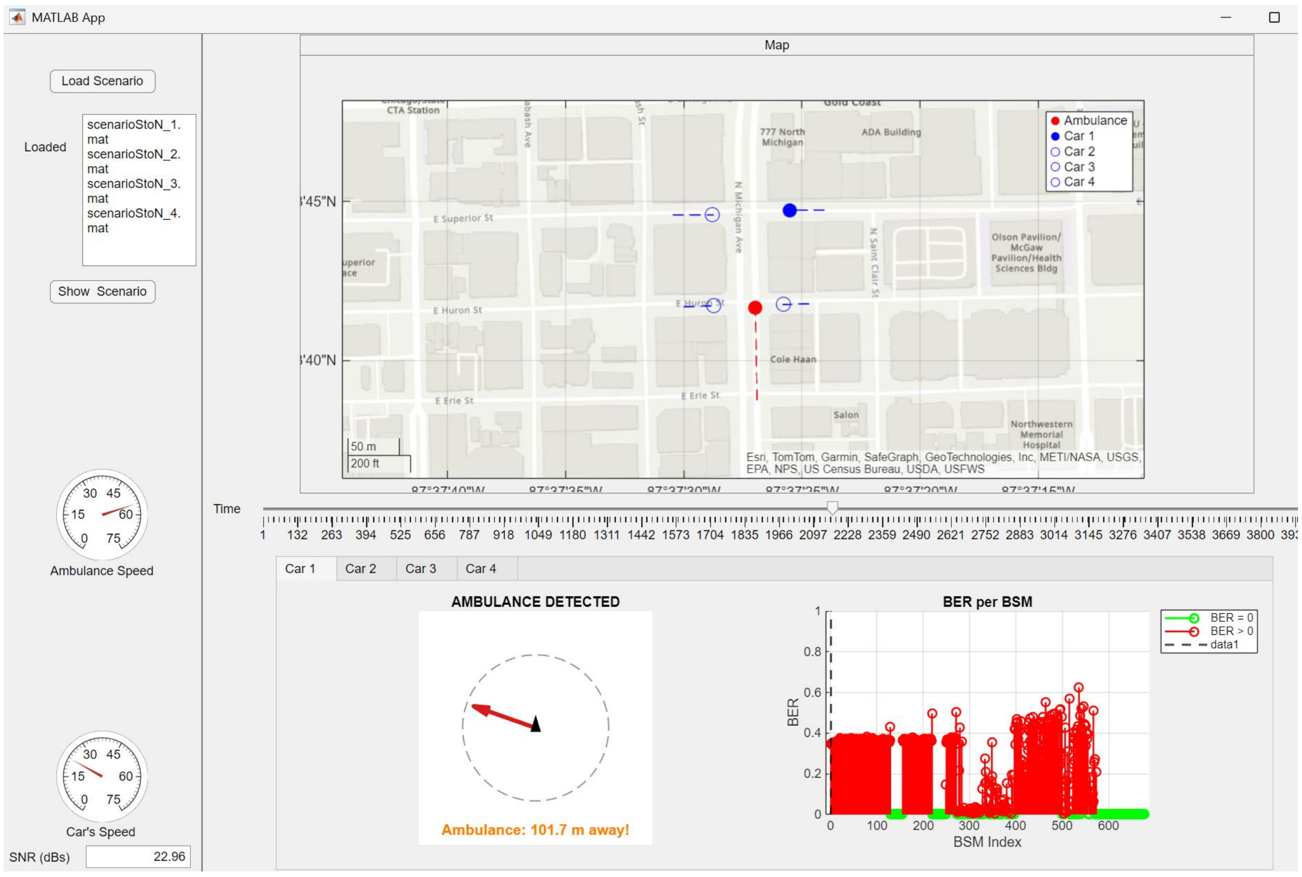


Figure 21. Results viewer App

In the above example, Car number 1 (selected in blue), is detecting the ambulance (in red) from 100 meters away, with a Signal to Noise Ratio of 22 dB. Since the car is moving West, the system points towards the ambulance with a red arrow pointing at around 75º to the left of the car, matching the real point of view of the driver.

If the time slider is moved forwards, the App shows the following result:

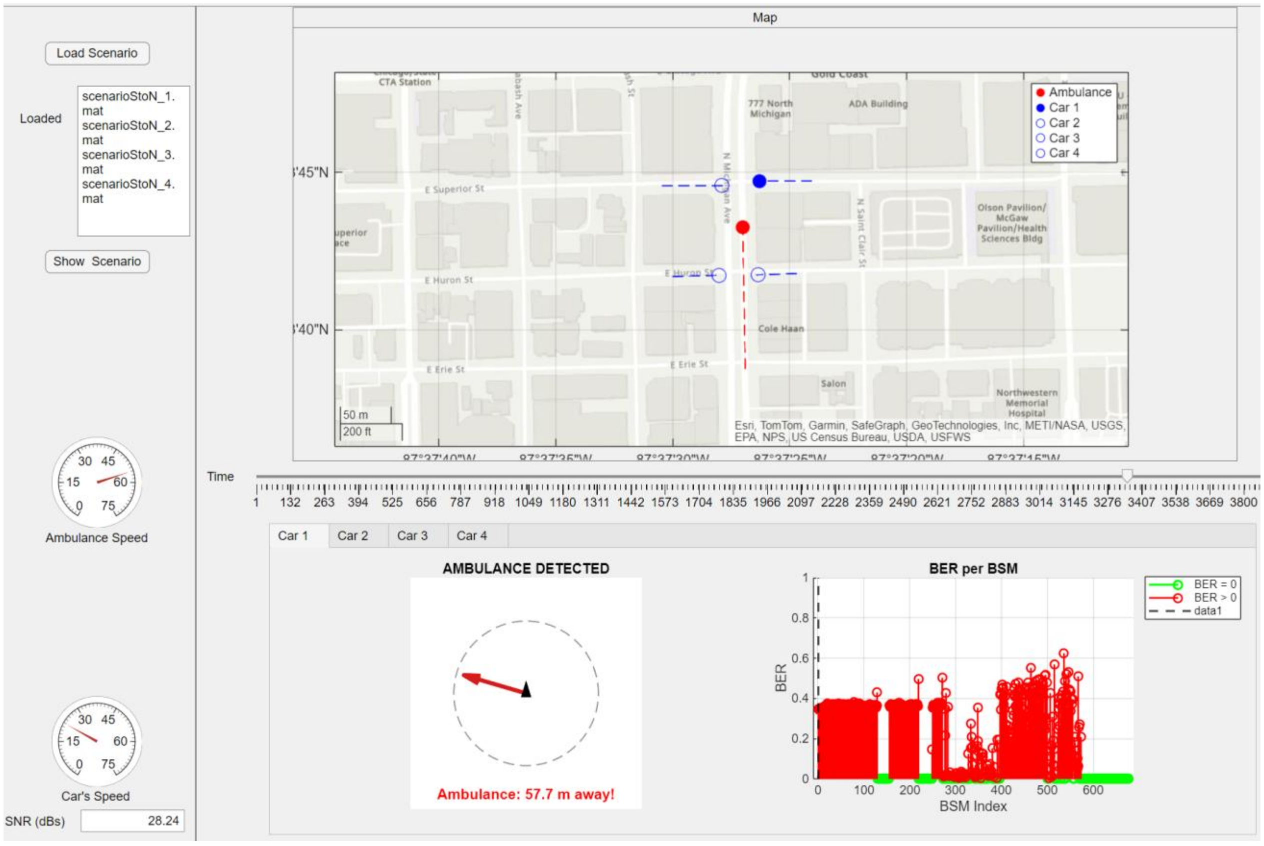


Figure 22. Results viewer app continuation

As shown in Figure 22, now the ambulance is much closer to Car 1 (around 57 meters away), and the direction of the arrow has also slightly changed, so that it is still pointing at the ambulance. Because the vehicles are closer together, the SNR is now higher (28,24 dB).

By pressing the Show Scenario button, a new window shows the signal propagation in 3D between the ambulance and the selected car, with the multiple paths that are being calculated in the background (Figure 23).

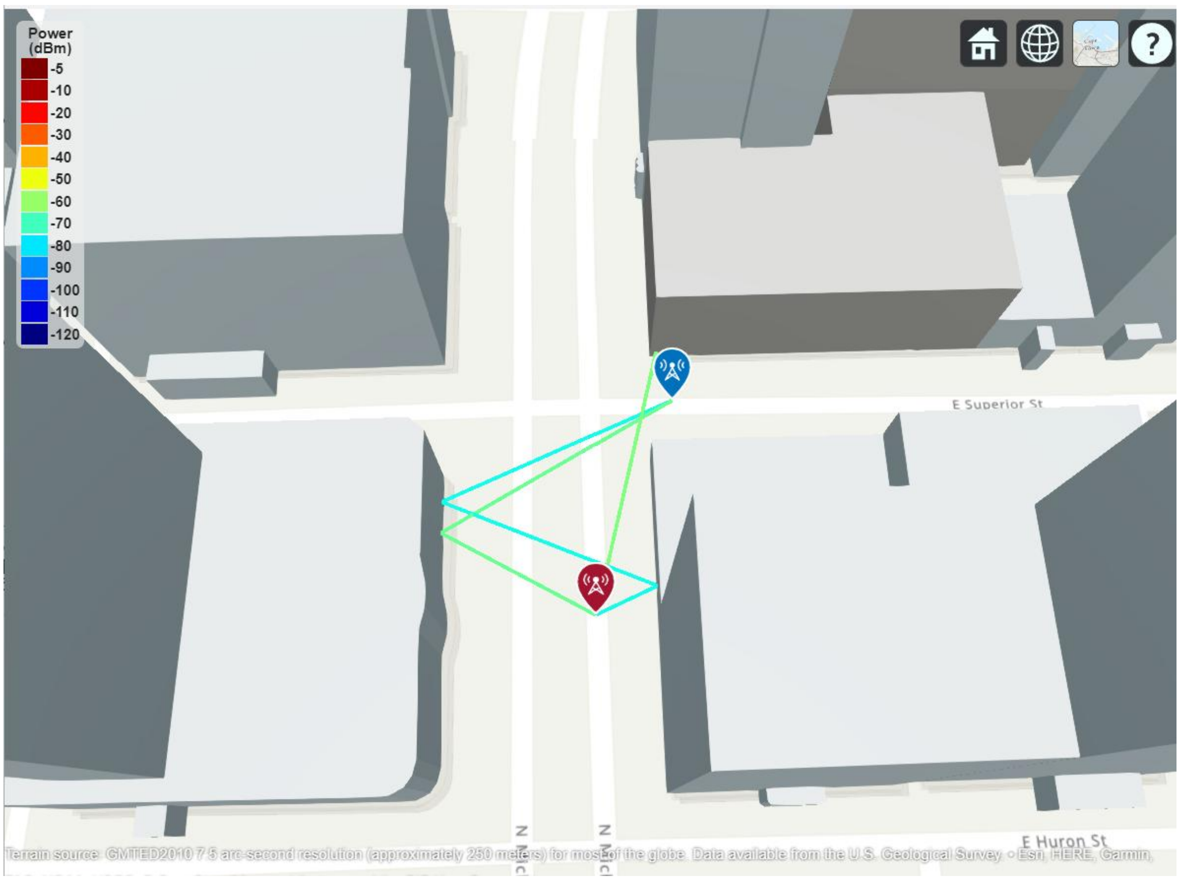

Power (dBm)
-5
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
E Superior St
E Huron St
N Mic
N Mic
Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Esri, HERE, Garmin,

Figure 23. 3D Scenario result

Thanks to this system, the driver of Car 1 would have been successfully alerted of the incoming ambulance.

This app transforms raw simulation data into an intuitive, driver-oriented display, bridging the gap between communication performance metrics and the real-world experience of an emergency vehicle detection system.

The entire code that runs this app is copied here:

```
classdef app2 < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                    matlab.ui.Figure
        GridLayout                  matlab.ui.container.GridLayout
        LeftPanel                   matlab.ui.container.Panel
        SNRdBsEditField             matlab.ui.control.NumericEditField
        SNRdBsEditFieldLabel      matlab.ui.control.Label
        CarsSpeedGauge            matlab.ui.control.Gauge
        CarsSpeedGaugeLabel       matlab.ui.control.Label
        AmbulanceSpeedGauge       matlab.ui.control.Gauge
        AmbulanceSpeedGaugeLabel  matlab.ui.control.Label
        ShowScenarioButton        matlab.ui.control.Button
        LoadedTextArea            matlab.ui.control.TextArea
        LoadedTextAreaLabel       matlab.ui.control.Label
        LoadScenarioButton        matlab.ui.control.Button
        RightPanel                matlab.ui.container.Panel
        TabGroup                  matlab.ui.container.TabGroup
        Tab                       matlab.ui.container.Tab
        MapPanel                  matlab.ui.container.Panel
        TimeSlider                matlab.ui.control.Slider
        TimeSliderLabel           matlab.ui.control.Label
    end

    % Properties that correspond to apps with auto-reflow
    properties (Access = private)
        onePanelWidth = 576;
    end


    % Public properties that correspond to the Simulink model
    properties (Access = public, Transient)
        Simulation simulink.Simulation
    end


    properties (Access = private)
        fileNames;
        pathName;
        scenarioList;
        gx;
        timeIdx;
        ambulancePointHandle;
        carPointHandles;
        ambulance_lats_rx;
        ambulance_longs_rx;
        car_headings;
        d = 1;
        n = 3;
        d_max = 300;
        ambulance_coords_filtered;
        car_outputs = [];
        DashboardAxes   % Stores the UIAxes handle for each car tab
        BERAxes %Stores the UIAxes for the BER plots for each car
        colors
        selectedCar
        cars
        viewer
    end
    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: LoadScenarioButton
        function LoadScenarioButtonPushed(app, event)
            % Reset variables
            app.scenarioList = [];
            app.timeIdx = 1;
            app.TimeSlider.Value = 1;

            % Reset data containers
            app.ambulance_lats_rx = [];
            app.ambulance_longs_rx = [];
            app.ambulance_coords_filtered = {};
            app.car_outputs = [];
            app.car_headings = [];
            app.selectedCar = "Car 1";

            % Reset n to its base value (assuming default n = 3)
            app.n = 3;

            if ~isempty(app.gx) && isvalid(app.gx)
                delete(app.gx);
            end

            [app.fileNames, app.pathName] = uigetfile('*.mat', 'Select scenario
files', 'MultiSelect', 'on');
            fprintf('Loaded Scenarios: %d', length(app.fileNames));
            if iscell(app.fileNames)
                for k = 1:length(app.fileNames)
                    app.scenarioList{k} = load_scenario(fullfile(app.pathName,
app.fileNames{k}));
                end
            else
                app.scenarioList{1} = load_scenario(fullfile(app.pathName,
app.fileNames));
            end
            app.LoadedTextArea.Value = app.fileNames;

            app.colors = lines(length(app.scenarioList));  % Generate distinct
colors

            %app.gx = geoaxes(app.RightPanel);
            app.gx = geoaxes('Parent', app.MapPanel, ...
                'Units', 'normalized', ...
                'Position', [0 0 1 1]);  % Fill entire panel responsively

            %set(0, 'CurrentFigure', app.UIFigure)       % Make fig current
            %gx = geoaxes(app.RightPanel);
            %gx.HandleVisibility
            hold(app.gx, "on")
            geobasemap(app.gx, "topographic")
            lat = [40 -74]; % Example latitudes
            lon = [38 -72]; % Example longitudes
            geoplot(app.gx, lat, lon, 'o', 'HandleVisibility','off');
            geolimits(app.gx, app.scenarioList{1}.lat_limits,
app.scenarioList{1}.lon_limits);

            %geoplot(app.gx, app.scenarioList{1}.ambulance_lats_tx(1),
app.scenarioList{1}.ambulance_longs_tx(1), 'ro');
            % Plot current ambulance position
            app.ambulancePointHandle = geoscatter(app.gx, ...
                app.scenarioList{1}.ambulance_lats_tx(1), ...
                app.scenarioList{1}.ambulance_longs_tx(1), ...
                100, 'r', 'o', 'filled', 'DisplayName', 'Ambulance');

            %geoplot(app.gx, app.scenarioList{1}.ambulance_lats_tx,
app.scenarioList{1}.ambulance_longs_tx, 'r--');
            % Plot Ambulance Trajectory
            %geoplot(app.gx, ...
                %app.scenarioList{1}.ambulance_lats_tx, ...
                %app.scenarioList{1}.ambulance_longs_tx, ...
                %'r--', 'LineWidth', 1, 'HandleVisibility', 'off');
            app.TimeSlider.Limits = [1,
length(app.scenarioList{1}.ambulance_lats_tx)];
            app.n = app.n*app.scenarioList{1}.nSym_per_BSM;


            for k = 1:length(app.scenarioList)
                %geoplot(app.gx, app.scenarioList{k}.car_lats(1),
app.scenarioList{k}.car_longs(1), 'bo');

                if app.selectedCar == sprintf("Car %d", k)
                    app.carPointHandles{k} = geoscatter(app.gx, ...
                        app.scenarioList{k}.car_lats(1), ...
                        app.scenarioList{k}.car_longs(1), ...
                        100, 'b', 'o', 'filled', 'DisplayName', sprintf('Car %d',
k));

                else
                    app.carPointHandles{k} = geoscatter(app.gx, ...
                        app.scenarioList{k}.car_lats(1), ...
                        app.scenarioList{k}.car_longs(1), ...
                        100, 'b', 'o', 'DisplayName', sprintf('Car %d', k));

                end

                %geoplot(app.gx, app.scenarioList{k}.car_lats,
app.scenarioList{k}.car_longs, 'b--');
                %geoplot(app.gx, ...
                    %app.scenarioList{k}.car_lats, ...
                    %app.scenarioList{k}.car_longs, ...
                    %'--', 'Color', 'b', 'LineWidth', 1, ...
                    %'HandleVisibility', 'off');
                app.ambulance_lats_rx(:, k) =
app.scenarioList{k}.ambulance_lats_rx;
                app.ambulance_longs_rx(:, k) =
app.scenarioList{k}.ambulance_longs_rx;
                app.car_headings(k) =
calculate_heading(app.scenarioList{k}.cars_orig, app.scenarioList{k}.cars_dest);

            end

            legend(app.gx);

            % Clear previous tabs if any
            delete(app.TabGroup.Children);

            numCars = length(app.scenarioList);
            app.DashboardAxes = gobjects(1, numCars);  % Initialize axes handles
            app.BERAxes = gobjects(1, numCars);

            for k = 1:numCars
                app.cars(k) = sprintf("Car %d", k);
                % Create a new tab for each car
                tab = uitab(app.TabGroup);
                tab.Title = sprintf("Car %d", k);

                % Create dashboard UIAxes
                dash_ax = uiaxes(tab);
                dash_ax.Position = [20 20 220 140];
                title(dash_ax, sprintf('Car %d Dashboard', k));
                dash_ax.XTick = [];
                dash_ax.YTick = [];
                dash_ax.XColor = 'none';
                dash_ax.YColor = 'none';
                axis(dash_ax, 'equal');
                app.DashboardAxes(k) = dash_ax;

                % Create BER UIAxes next to dashboard
                ber_ax = uiaxes(tab);
                ber_ax.Position = [250 20 220 140];  % Adjust spacing/size as
needed
                title(ber_ax, 'BER per BSM');
                xlabel(ber_ax, 'BSM Index');
                ylabel(ber_ax, 'BER');
                grid(ber_ax, 'on');
                app.BERAxes(k) = ber_ax;


                plot_BER(app.BERAxes(k), app.scenarioList{k}.BER_vec,
app.timeIdx);

            end

            app.viewer = siteviewer(Buildings=app.scenarioList{1}.map,
Basemap="topographic");
```

```
            app.AmbulanceSpeedGauge.Value =
app.scenarioList{1}.ambulance_speed*2.236;
            app.CarsSpeedGauge.Value = app.scenarioList{1}.car_speed*2.236;


        end

        % Value changed function: TimeSlider
        function TimeSliderValueChanged(app, event)
            value = app.TimeSlider.Value;
            app.timeIdx = round(value);
            app.TimeSlider.Value = app.timeIdx;

            % Remove previous plotted points if they exist
            if ~isempty(app.ambulancePointHandle) &&
isvalid(app.ambulancePointHandle)
                delete(app.ambulancePointHandle);
            end
            if ~isempty(app.carPointHandles)
                for k = 1:length(app.carPointHandles)
                    if isvalid(app.carPointHandles{k})
                        delete(app.carPointHandles{k});
                    end
                end
            end

            % Plot new ambulance point
            %app.ambulancePointHandle = geoplot(app.gx, ...
                %app.scenarioList{1}.ambulance_lats_tx(app.timeIdx), ...
                %app.scenarioList{1}.ambulance_longs_tx(app.timeIdx), ...
                %'ro', 'MarkerSize', 8, 'DisplayName', 'Ambulance');

            app.ambulancePointHandle = geoscatter(app.gx, ...
                app.scenarioList{1}.ambulance_lats_tx(app.timeIdx), ...
                app.scenarioList{1}.ambulance_longs_tx(app.timeIdx), ...
                100, 'r', 'o', 'filled', 'DisplayName', 'Ambulance');

            % Plot Ambulance Trajectory
            geoplot(app.gx, ...
                app.scenarioList{1}.ambulance_lats_tx(1:app.timeIdx), ...
                app.scenarioList{1}.ambulance_longs_tx(1:app.timeIdx), ...
                'r--', 'LineWidth', 1, 'HandleVisibility', 'off');

            % Plot new car positions
            app.carPointHandles = cell(1, length(app.scenarioList));
            for k = 1:length(app.scenarioList)
                %app.carPointHandles{k} = geoplot(app.gx, ...
                    %app.scenarioList{k}.car_lats(app.timeIdx), ...
                    %app.scenarioList{k}.car_longs(app.timeIdx), ...
                    %'bo', 'MarkerSize', 8, 'DisplayName', sprintf('Car %d', k));

                if app.selectedCar == sprintf("Car %d", k)
                    app.carPointHandles{k} = geoscatter(app.gx, ...
                        app.scenarioList{k}.car_lats(app.timeIdx), ...
                        app.scenarioList{k}.car_longs(app.timeIdx), ...
                        100, 'b', 'o', 'filled', 'DisplayName', sprintf('Car %d',
k));

                    app.SNRdBsEditField.Value =
app.scenarioList{k}.SNR_dBs(app.timeIdx);


                else
                    app.carPointHandles{k} = geoscatter(app.gx, ...
                        app.scenarioList{k}.car_lats(app.timeIdx), ...
                        app.scenarioList{k}.car_longs(app.timeIdx), ...
                        100, 'b', 'o', 'DisplayName', sprintf('Car %d', k));

                end

                geoplot(app.gx, ...
                    app.scenarioList{k}.car_lats(1:app.timeIdx), ...
                    app.scenarioList{k}.car_longs(1:app.timeIdx), ...
                    '--', 'Color', 'b', 'LineWidth', 1, ...
                    'HandleVisibility', 'off');

                car_coords = [app.scenarioList{k}.car_lats',
app.scenarioList{k}.car_longs'];
                ambulance_coords =
[app.scenarioList{k}.ambulance_lats_rx(1:app.timeIdx)',
app.scenarioList{k}.ambulance_longs_rx(1:app.timeIdx)'];
                app.ambulance_coords_filtered{k} = filter_rx(ambulance_coords,
app.d, app.n, car_coords,app.d_max);

                car_coords_idx = [app.scenarioList{k}.car_lats(app.timeIdx),
app.scenarioList{k}.car_longs(app.timeIdx)];
                [output_dir, output_dist] =
calculate_output(app.ambulance_coords_filtered{k}(app.timeIdx,:), car_coords_idx,
app.car_headings(k));
                app.car_outputs(k,:) = [output_dir, output_dist];

                % Normalize direction to be relative to car heading (0 = front)
                %relative_dir = mod(output_dir - app.car_headings(k), 360);

                % Plot dashboard arrow
                plot_dashboard(app.DashboardAxes(k), output_dir, output_dist);

            end





        end
        % Callback function
        function ShowTrajectoriesCheckBoxValueChanged(app, event)
            value = app.ShowTrajectoriesCheckBox.Value;
            app.plot_trajectory = value;
        end

        % Selection change function: TabGroup
        function TabGroupSelectionChanged(app, event)
            selectedTab = app.TabGroup.SelectedTab;
            app.selectedCar = selectedTab.Title;

            if ~isempty(app.carPointHandles)
                for k = 1:length(app.carPointHandles)
                    if isvalid(app.carPointHandles{k})
                        delete(app.carPointHandles{k});
                    end
                end
            end

            % Plot new car positions
            app.carPointHandles = cell(1, length(app.scenarioList));
            for k = 1:length(app.scenarioList)
                %app.carPointHandles{k} = geoplot(app.gx, ...
                    %app.scenarioList{k}.car_lats(app.timeIdx), ...
                    %app.scenarioList{k}.car_longs(app.timeIdx), ...
                    %'bo', 'MarkerSize', 8, 'DisplayName', sprintf('Car %d', k));

                if app.selectedCar == sprintf("Car %d", k)
                    app.carPointHandles{k} = geoscatter(app.gx, ...
                        app.scenarioList{k}.car_lats(app.timeIdx), ...
                        app.scenarioList{k}.car_longs(app.timeIdx), ...
                        100, 'b', 'o', 'filled', 'DisplayName', sprintf('Car %d',
k));
                    app.CarsSpeedGauge.Value =
app.scenarioList{k}.car_speed*2.236;
                    app.SNRdBsEditField.Value =
app.scenarioList{k}.SNR_dBs(app.timeIdx);



                else
                    app.carPointHandles{k} = geoscatter(app.gx, ...
                        app.scenarioList{k}.car_lats(app.timeIdx), ...
                        app.scenarioList{k}.car_longs(app.timeIdx), ...
                        100, 'b', 'o', 'DisplayName', sprintf('Car %d', k));

                end
            end

        end

        % Button pushed function: ShowScenarioButton
        function ShowScenarioButtonPushed(app, event)
```

```
            % -- Ray Tracing Model Setup --
            rtpm = propagationModel("raytracing", ...
                Method="sbr", ...
                MaxNumReflections=app.scenarioList{1}.max_number_reflections, ...
                BuildingsMaterial="concrete", ...
                TerrainMaterial="concrete");

            for k = 1:length(app.scenarioList)
                if app.selectedCar == sprintf("Car %d", k)
                    tx_lat = app.scenarioList{k}.ambulance_lats_tx(app.timeIdx);
                    tx_long =
app.scenarioList{k}.ambulance_longs_tx(app.timeIdx);
                    rx_lat = app.scenarioList{k}.car_lats(app.timeIdx);
                    rx_long = app.scenarioList{k}.car_longs(app.timeIdx);

                    % Setup txsite and rxsite for ray tracing
                    tx = txsite(Name="Ambulance", ...
                        Latitude=tx_lat, ...
                        Longitude=tx_long, ...
                        AntennaHeight=3, ...
                        TransmitterPower=app.scenarioList{1}.tx_power, ...
                        TransmitterFrequency=app.scenarioList{1}.f);

                    rx = rxsite(Name=sprintf("Car %d", k), ...
                        Latitude=rx_lat, ...
                        Longitude=rx_long, ...
                        AntennaHeight=1);

                    clearMap(app.viewer)
                    show(tx)
                    raytrace(tx,rx,rtpm)
                end
            end

        end

        % Changes arrangement of the app based on UIFigure width
        function updateAppLayout(app, event)
            currentFigureWidth = app.UIFigure.Position(3);
            if(currentFigureWidth <= app.onePanelWidth)
                % Change to a 2x1 grid
                app.GridLayout.RowHeight = {480, 480};
                app.GridLayout.ColumnWidth = {'1x'};
                app.RightPanel.Layout.Row = 2;
                app.RightPanel.Layout.Column = 1;
            else
                % Change to a 1x2 grid
                app.GridLayout.RowHeight = {'1x'};
                app.GridLayout.ColumnWidth = {190, '1x'};
                app.RightPanel.Layout.Row = 1;
                app.RightPanel.Layout.Column = 2;
            end
        end
```

```
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.AutoResizeChildren = 'off';
            app.UIFigure.Position = [100 100 640 480];
            app.UIFigure.Name = 'MATLAB App';
            app.UIFigure.SizeChangedFcn = createCallbackFcn(app,
@updateAppLayout, true);

            % Create GridLayout
            app.GridLayout = uigridlayout(app.UIFigure);
            app.GridLayout.ColumnWidth = {190, '1x'};
            app.GridLayout.RowHeight = {'1x'};
            app.GridLayout.ColumnSpacing = 0;
            app.GridLayout.RowSpacing = 0;
            app.GridLayout.Padding = [0 0 0 0];
            app.GridLayout.Scrollable = 'on';

            % Create LeftPanel
            app.LeftPanel = uipanel(app.GridLayout);
            app.LeftPanel.Layout.Row = 1;
            app.LeftPanel.Layout.Column = 1;

            % Create LoadScenarioButton
            app.LoadScenarioButton = uibutton(app.LeftPanel, 'push');
            app.LoadScenarioButton.ButtonPushedFcn = createCallbackFcn(app,
@LoadScenarioButtonPushed, true);
            app.LoadScenarioButton.Position = [46 424 100 22];
            app.LoadScenarioButton.Text = 'Load Scenario';

            % Create LoadedTextAreaLabel
            app.LoadedTextAreaLabel = uilabel(app.LeftPanel);
            app.LoadedTextAreaLabel.HorizontalAlignment = 'right';
            app.LoadedTextAreaLabel.Position = [7 344 55 58];
            app.LoadedTextAreaLabel.Text = 'Loaded';

            % Create LoadedTextArea
            app.LoadedTextArea = uitextarea(app.LeftPanel);
            app.LoadedTextArea.Position = [77 344 108 60];

            % Create ShowScenarioButton
            app.ShowScenarioButton = uibutton(app.LeftPanel, 'push');
            app.ShowScenarioButton.ButtonPushedFcn = createCallbackFcn(app,
@ShowScenarioButtonPushed, true);
            app.ShowScenarioButton.Position = [46 309 100 22];
            app.ShowScenarioButton.Text = 'Show  Scenario';
            % Create AmbulanceSpeedGaugeLabel
            app.AmbulanceSpeedGaugeLabel = uilabel(app.LeftPanel);
            app.AmbulanceSpeedGaugeLabel.HorizontalAlignment = 'center';
            app.AmbulanceSpeedGaugeLabel.Position = [44 168 103 22];
            app.AmbulanceSpeedGaugeLabel.Text = 'Ambulance Speed';

            % Create AmbulanceSpeedGauge
            app.AmbulanceSpeedGauge = uigauge(app.LeftPanel, 'circular');
            app.AmbulanceSpeedGauge.Limits = [0 75];
            app.AmbulanceSpeedGauge.MajorTicks = [0 15 30 45 60 75];
            app.AmbulanceSpeedGauge.Position = [52 188 87 87];

            % Create CarsSpeedGaugeLabel
            app.CarsSpeedGaugeLabel = uilabel(app.LeftPanel);
            app.CarsSpeedGaugeLabel.HorizontalAlignment = 'center';
            app.CarsSpeedGaugeLabel.Position = [59 43 71 22];
            app.CarsSpeedGaugeLabel.Text = 'Car''s Speed';

            % Create CarsSpeedGauge
            app.CarsSpeedGauge = uigauge(app.LeftPanel, 'circular');
            app.CarsSpeedGauge.Limits = [0 75];
            app.CarsSpeedGauge.MajorTicks = [0 15 30 45 60 75];
            app.CarsSpeedGauge.Position = [52 63 87 87];

            % Create SNRdBsEditFieldLabel
            app.SNRdBsEditFieldLabel = uilabel(app.LeftPanel);
            app.SNRdBsEditFieldLabel.HorizontalAlignment = 'right';
            app.SNRdBsEditFieldLabel.Position = [3 19 62 22];
            app.SNRdBsEditFieldLabel.Text = 'SNR (dBs)';

            % Create SNRdBsEditField
            app.SNRdBsEditField = uieditfield(app.LeftPanel, 'numeric');
            app.SNRdBsEditField.Position = [80 19 100 22];

            % Create RightPanel
            app.RightPanel = uipanel(app.GridLayout);
            app.RightPanel.Layout.Row = 1;
            app.RightPanel.Layout.Column = 2;

            % Create TimeSliderLabel
            app.TimeSliderLabel = uilabel(app.RightPanel);
            app.TimeSliderLabel.HorizontalAlignment = 'right';
            app.TimeSliderLabel.Position = [6 232 31 22];
            app.TimeSliderLabel.Text = 'Time';

            % Create TimeSlider
            app.TimeSlider = uislider(app.RightPanel);
            app.TimeSlider.Limits = [0 200];
            app.TimeSlider.ValueChangedFcn = createCallbackFcn(app,
@TimeSliderValueChanged, true);
            app.TimeSlider.Position = [58 241 374 3];
            % Create MapPanel
            app.MapPanel = uipanel(app.RightPanel);
            app.MapPanel.TitlePosition = 'centertop';
            app.MapPanel.Title = 'Map';
            app.MapPanel.Position = [93 257 268 222];

            % Create TabGroup
            app.TabGroup = uitabgroup(app.RightPanel);
            app.TabGroup.SelectionChangedFcn = createCallbackFcn(app,
@TabGroupSelectionChanged, true);
            app.TabGroup.Position = [70 17 309 181];

            % Create Tab
            app.Tab = uitab(app.TabGroup);
            app.Tab.Title = 'Tab';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = app2

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

Chapter 6. RESULTS

With the program described in Chapter 5, this section summarizes the obtained results.

The implementation and testing of the proposed V2V communication-based ambulance alert system have yielded results that fully align with the objectives and motivations established at the beginning of this project. The system was designed with the primary goal of providing drivers with timely, reliable alerts about incoming emergency vehicles, even under challenging channel conditions, so that they can take the necessary action to yield and facilitate emergency response.

Extensive field-scenario simulations, combined with controlled testing, have demonstrated that the system can successfully detect and alert drivers in real time under a wide range of conditions. The most significant outcomes are summarized and discussed below.

6.1 MAXIMUM EFFECTIVE RANGE AND SNR REQUIREMENTS

After rigorous testing across multiple environments, the system has shown the capability to alert drivers to an approaching ambulance at distances of up to 200 meters when the Signal-to-Noise Ratio (SNR) is at least 8 dB. This range represents the upper bound of reliable detection and provides more than enough time for drivers to react in urban or suburban traffic conditions.

6.2 GUARANTEED ALERT RADIUS

One of the project's key goals was to ensure that, within a certain proximity to the ambulance, every driver would receive a reliable alert without exception. Testing in diverse scenarios (including urban canyons, open suburban roads, and intersections with heavy multipath) confirmed that:

- Within a 100-meter radius, the system alerted 100% of drivers in every single test case.
- Within a 150-meter radius, the success rate was 90%, a level of reliability still considered excellent for real-world deployment.

These results directly address the motivation of enhancing road safety by ensuring that no driver within the critical danger zone is left unaware of an approaching emergency vehicle.

6.3 CHANNEL IMPAIRMENT COMPENSATION

The system's signal processing chain (including channel estimation, iCDP-based equalization, and robust BSM decoding) was able to fully compensate for channel distortions caused by:

- Multipath propagation from surrounding buildings and vehicles.
- Doppler shifts due to high-speed relative motion between vehicles.
- Additive channel noise under various SNR levels.

This 100% compensation rate within the 100-meter guaranteed alert radius ensures that the core BSM data (particularly latitude and longitude) remains accurate enough for the trajectory-based ambulance detection logic to function without failure.

6.4 PERFORMANCE AT HIGH SPEEDS

The system has been validated for vehicle speeds up to 90 mph (145 km/h), covering typical urban, suburban, and even certain highway scenarios. At these speeds, the system maintained its stated detection ranges and reliability rates, demonstrating robustness against high Doppler conditions that might otherwise degrade performance in V2V systems.

6.5 TRANSMITTER POWER CONSIDERATIONS

All tests were conducted using a transmit power of 1 W, which was sufficient to achieve the desired performance metrics. Increasing transmit power was tested but did not produce a significant improvement in detection range or reliability. This is explained by the fact that multipath ray power scales with transmit power, meaning that while the direct path gains strength, so do the interfering reflections. This finding reinforces the conclusion that signal processing techniques ,rather than brute-force transmission power, are the key enabler of the system's performance.

6.6 ACHIEVEMENT OF PROJECT OBJECTIVES

The results confirm that the system fully delivers on the vision set at the start of the project: improving road safety by ensuring drivers receive timely and reliable alerts of an approaching ambulance, even in challenging environments. By leveraging V2V communications, the system overcomes the limitations of traditional audible sirens, which can be masked by traffic noise or blocked by buildings in urban settings. It consistently provides alerts within a guaranteed 100-meter range, often extending up to 200 meters, giving drivers sufficient time to react and yield. The robust performance under multipath, Doppler, and noise conditions, combined with successful operation at speeds up to 90 mph, demonstrates that the system is both practical and resilient for real-world deployment.

Chapter 7. SYSTEM LIMITATIONS

While the proposed system has demonstrated strong performance in simulation, several limitations must be acknowledged before considering real-world deployment.

The simulation environment models the urban landscape with buildings and static terrain features, enabling accurate estimation of multipath effects caused by reflections off large structures. However, it does not model reflections and diffractions from other vehicles, street furniture, or dynamic elements in the scene. In dense traffic, the presence of multiple nearby cars, buses, or metallic objects can increase the complexity of the multipath environment, introducing additional variations in the channel response. In theory, the implemented iCDP (improved Channel Distortion Prediction) algorithm should be able to handle this, as it does not depend on the absolute level of distortion but rather on tracking the relative changes between consecutive symbols. Still, this behavior has not been verified under such conditions, and experimental validation in real-world traffic would be necessary to confirm its robustness.

The system's channel model also incorporates additive white Gaussian noise (AWGN) and noise caused by atmospheric phenomena such as rain. However, it does not simulate interference from other transmitters operating in the same frequency range. While the 5.9 GHz ITS (Intelligent Transportation Systems) band is relatively clean and unlikely to be heavily congested, especially for safety-critical applications, in real deployments it is possible that other devices or misconfigured equipment could introduce interference. Although this is expected to be a rare occurrence, it remains a factor to consider for future versions of the system.

Finally, it is important to note that the entire project is based on simulated physics. The ray tracing models, noise profiles, and mobility patterns are designed to approximate reality as closely as possible, and the results obtained align with what would be expected from a real-world system. Nevertheless, physical measurements often reveal subtleties that simulations cannot fully capture, such as small-scale fading due to irregular structures, hardware imperfections, or unmodeled environmental factors. As such, while the presented results are promising, they cannot be considered definitive until validated with real-world field tests.

Chapter 8. REAL WORLD IMPLEMENTATION

Translating this project from a simulation-based prototype into a functioning real-world system would require both hardware deployment and coordinated action between multiple stakeholders. On the hardware side, passenger vehicles would need to incorporate a 5.9 GHz V2X on-board unit capable of receiving BSM messages, together with a low-profile external antenna and a secure element to store the necessary communication credentials. This hardware would interface with the car's existing electronics to process the received BSMs, determine the relative direction and distance of the ambulance, and display a visual and audio alert through the dashboard or infotainment system. Since most modern vehicles already contain GNSS receivers, computing platforms, and display interfaces, the main additions would be the dedicated V2X radio, antenna, and secure module, along with the integration effort to connect these elements to the vehicle's software and human–machine interface.

Emergency vehicles would require a similar V2X on-board unit, but with transmission enabled. This unit would also be paired with a multi-band vehicular antenna and a control interface allowing ambulance crews to start or stop broadcasts, ideally linked directly to the siren and lights so that messages are automatically sent when the vehicle is responding to an emergency. The transmitted BSMs would be digitally signed using agency-managed credentials that identify the sender as an authorized emergency vehicle without revealing unnecessary identifying information. Integration into the emergency vehicle's operating procedures would be straightforward: crews would only need to verify that the system is active during emergency runs.

Beyond the hardware, a real-world rollout would require addressing several regulatory, legal, and interoperability considerations. All radios would need to comply with national spectrum allocations for the 5.9 GHz ITS band and pass automotive EMC and safety certifications. A public key infrastructure (PKI) would be required to issue, manage, and revoke digital certificates for both emergency vehicles and private cars, ensuring that only authorized units can send alerts. Privacy policies would need to clarify how pseudonymous certificates are rotated and how location data is handled, preventing any long-term tracking of vehicles. Liability considerations would also need to be addressed, making clear that the alert is advisory and that drivers remain responsible for their own actions.

For car manufacturers, integrating this feature would involve sourcing and installing the V2X hardware, adapting vehicle software to process BSMs using the algorithms developed in this project, and ensuring the alert display is consistent with existing user interface guidelines. They would also have to include the feature in their cybersecurity maintenance programs and ensure compatibility with OTA updates. For customers purchasing these vehicles, the feature would come pre-installed, requiring no special actions beyond acknowledging its presence and understanding what the alert means. Emergency services, in turn, would need to invest in transmit-capable units, integrate them into their fleets, manage their cryptographic credentials, and provide minimal training to ensure correct use.

The societal implications of such an implementation would be significant. Drivers would gain early awareness of approaching emergency vehicles, potentially reducing response times and improving safety for both responders and other road users. For municipalities and transport authorities, the system would represent a low-infrastructure investment, as it relies entirely on vehicle-to-vehicle communication rather than roadside installations. The main challenge would be ensuring consistent standards across manufacturers and regions, so that all equipped vehicles can communicate seamlessly.

The path to deployment would begin with the formation of a pilot program involving one or more car manufacturers, a willing ambulance service, a V2X hardware supplier, and a municipal transportation authority. A small fleet of equipped passenger cars and ambulances could be tested first in controlled environments such as closed tracks or specific urban corridors. These tests would validate the system's performance under real-world multipath and interference conditions, refine the filtering algorithms for discarding implausible coordinates, and assess the human–machine interface for clarity and minimal distraction. From there, progressively larger trials could be conducted in live traffic, with the eventual goal of scaling to full-fleet adoption once technical, regulatory, and interoperability hurdles are cleared.

The costs for such an implementation, as discussed earlier, would mainly stem from the additional V2X modules, antennas, and secure storage hardware, together with integration and installation labor. Because modern vehicles already carry much of the supporting technology, the incremental cost per vehicle would be modest, especially when produced at scale. For emergency services, the investment would be similar per vehicle, with additional expenses for managing credentials and integrating the control interface. In the longer term, once these components are incorporated into factory builds, the marginal cost of adding the capability to each new car or ambulance would drop significantly, making wide deployment economically feasible.

Chapter 9. REFERENCES

1. AutoTalks. (n.d.). *AutoTalks DSRC vs C-V2X*. Retrieved from https://auto-talks.com/technology/dsrc-vs-c-v2x/
2. Cheng, Z. a. (2014). *IEEE 802.11p for V2V Communications.*
3. Guillermo Francia, D. S. (2023). *Basic Safety Messages (BSM) Test Data Generation for Vehicle Security Machine Learning Systems.*
4. Kenney, J. B. (2020). *Dedicated Short Range Communications (DSRC) Standards in the United States.*
5. Mathworks. (n.d.). *comm.RayTracingChannel*. Retrieved from https://www.mathworks.com/help/comm/ref/comm.raytracingchannel-system-object.html
6. Mathworks. (n.d.). *Intersection Movement Assist Using Vehicle to Vehicle Communication*. Retrieved from https://www.mathworks.com/help/driving/ug/intersection-movement-assist-using-v2v.html
7. Mathworks. (s.f.). *OFDM Transmitter and Receiver with BPSK baseband, RF up-down conversion*. Obtenido de https://www.mathworks.com/matlabcentral/fileexchange/57494-ofdm-transmitter-and-receiver-with-bpsk-baseband-rf-up-down-conversion
8. Mathworks. (n.d.). *Raytracing.*
9. Mathworks. (n.d.). *Traffic Light Negotiation Using Vehicle-to-Everything Communication*. Retrieved from

https://www.mathworks.com/help/driving/ug/traffic-light-negotiation-using-v2x.html

10. SAE International. (2020). *SAE International J2735 Surfave Vehicle Standard.*

11. Tong Wang, A. H. (2018). *An improved channel estimation technique for IEEE 802.11p Standard in Vehicular Communications.*

ANEX: ALIGNMENT WITH SUSTAINABLE DEVELOPMENT GOALS (SDGS)

The project developed in this thesis, which focuses on the design and simulation of a vehicular communication system to detect emergency vehicles, aligns closely with several of the United Nations Sustainable Development Goals (SDGs). By improving road safety and enhancing the efficiency of emergency response, the project contributes to the advancement of key societal objectives.

First and foremost, the project directly supports SDG 3: Good Health and Well-Being. Traffic accidents involving ambulances often lead not only to injuries among road users but also to delays in medical attention for patients being transported. By enabling earlier detection of emergency vehicles and ensuring that other drivers can react in time, the system minimizes the risk of collisions and helps ambulances reach hospitals faster, ultimately improving health outcomes and saving lives.

The project also relates to SDG 9: Industry, Innovation, and Infrastructure, as it demonstrates the application of advanced wireless communication technologies in urban mobility. The use of OFDM modulation, channel equalization techniques, and the integration of intelligent vehicular communication systems represents an innovative approach to building smarter, more resilient transportation infrastructure.

Additionally, it contributes to SDG 11: Sustainable Cities and Communities. Safer and more efficient urban mobility systems are essential for sustainable cities. By reducing the risk of accidents and enhancing the coexistence between emergency services and everyday traffic, this system improves the safety and livability of urban environments, which is a key component of sustainable urban development.

Finally, the project supports SDG 16: Peace, Justice, and Strong Institutions, in the sense that it provides a framework for institutional collaboration between emergency services, vehicle manufacturers, and regulatory bodies. The adoption of such technologies requires coordination and regulatory frameworks that promote safety, trust, and efficiency in public services.

In conclusion, although this project is technical in nature, its impact extends far beyond engineering. By addressing a pressing issue in urban safety and emergency response, it directly contributes to the achievement of several Sustainable Development Goals, reinforcing the importance of technology as a driver of social progress.