

# Politecnico di Torino

Department of Electronics and Telecommunications



**Politecnico  
di Torino**

**Project: Software defined communication systems**

---

**802.11x Collision avoidance simulator**  
**REPORT**

---

<b>Student Name</b>	<b>Student ID</b>
Andrea Conese	324281
Andrea Paternò	331139
Hamze Ghorbani Koujani	325864
Umberto Pirovano	328012

**Professor:**  
Andrea Carena

**Tutor:**  
Carla Fabiana Chiasserini

**Assistant:**  
Corrado Puligheddu

# Contents

<b>1</b>	<b>Abstract</b>	<b>4</b>
<b>2</b>	<b>Organization</b>	<b>5</b>
<b>3</b>	<b>Version 0</b>	<b>6</b>
3.1	V0 achievements . . . . .	6
3.2	CV Model Selection . . . . .	7
<b>4</b>	<b>Version 1</b>	<b>8</b>
4.1	Wireless Channel Simulator based on IEEE 802.11ax . . . . .	8
4.2	Path Logic: Obstacle Detection and Decision Making . . . . .	10
4.3	Performance Analysis . . . . .	11
4.4	V1 Conclusion . . . . .	13
<b>5</b>	<b>Version 2</b>	<b>14</b>
5.1	GUI implementation . . . . .	14
5.2	Settings . . . . .	14
5.3	Scenario editor . . . . .	15
5.4	Simulation parameters . . . . .	15
5.5	Graphical results . . . . .	16
5.6	Output window . . . . .	17
5.7	Simulation example . . . . .	18
<b>6</b>	<b>Final comments and suggestions</b>	<b>21</b>
6.1	Extra Features . . . . .	21
6.2	Suggestions . . . . .	21
<b>7</b>	<b>User guide</b>	<b>22</b>
7.1	Docker . . . . .	22
7.2	Installation guide . . . . .	22
7.2.1	Supported Operative System . . . . .	22
7.2.2	AirSim . . . . .	22
7.2.3	Cloning the repository . . . . .	23
7.2.4	Dependencies . . . . .	23
7.3	Run the simulator . . . . .	24

## List of Figures

1	Gantt chart . . . . .	5
2	GitHub repository . . . . .	5
3	Average transmission time of the image vs distance (application layer) with shadowing component . . . . .	9
4	Average transmission time of the image vs distance (application layer) no shadowing component . . . . .	10
5	Speed trend . . . . .	11
6	Png, no compression . . . . .	12
7	Jpeg, quality: 80% . . . . .	12
8	GUI . . . . .	14
9	Settings . . . . .	14
10	Scenario editor . . . . .	15
11	Simulation parameters . . . . .	15
12	Overview: Last segmented image and speed trend . . . . .	16
13	Latency . . . . .	17
14	Output window . . . . .	17
15	Scenario and throttle . . . . .	18
16	Simulation example parameters . . . . .	18
17	Output window example . . . . .	19
18	Last segmented image and speed trend example . . . . .	19
19	Latency . . . . .	20

## List of Tables

1	Execution times for different CV models on various tasks and datasets. *Note: Execution times are based on processing a single image and may vary depending on hardware configuration (in this case a Nvidia RTX 2060 was used). .	7
2	Performance analysis of different CV models. . . . .	12

# 1 Abstract

This project aimed to develop a collision avoidance simulator for semi-autonomous vehicles in urban environments. The project utilized the AirSim simulator to instantiate a realistic urban driving environment. Images from the simulation were analyzed using computer vision models to detect obstacles and make decisions regarding speed and path planning for collision avoidance.

The project also included the simulation of a Wi-Fi 6 (IEEE 802.11ax) transmission channel to simulate the communication between the AirSim simulator and the edge service. This allowed for the transmission of images from AirSim to the computer vision model for obstacle detection and the transmission of speed updates back to AirSim for vehicle control.

The project was divided into three versions:

- **Version 0 (V0):** Focused on simulating the urban scenario in AirSim, preparing the data, and detecting obstacles in the image instances.
- **Version 1 (V1):** Implemented path control logic, including estimating the proximity of detected obstacles and providing information for speed changes. Additionally, a simulation of the Wi-Fi 6 channel was established to transmit images from AirSim to the computer vision model and sending instructions back to AirSim.
- **Version 2 (V2):** The final version of the project involved refining the simulator and implementing a graphical user interface (GUI) for user interaction.

Overall, this project successfully developed a collision avoidance simulator by combining simulation, computer vision, and communication technologies. The system's performance was evaluated using key performance indicators (KPIs) such as the latency. The project's outcome provides a useful and user-friendly tool that allows testing the effectiveness of an edge service based on wi-fi6 and Computer Vision in a vehicular scenario, where low latency is critical.

## 2 Organization

Three essential components of project management are timely delivery, efficiency, and order. To optimize these aspects, choices were made and an organizational plan with precise deadlines was developed and adhered to as faithfully as possible. All of this has been summarized in the **Gantt chart** below.

Initiative	Objective	April 1	April 2	April 3	April 4	May 1	May 2	May 3	May 4	June 1	June 2	June 3	June 4
Version 0	Simulate the urban scenario in <a href="#">AirSim</a> and achieve video frames.												
	Prepare the data and feed it to the CV model and detecting objects.												
	Basic results' analysis												
Version 1	Measure the distance from detected objects. Providing steering and speed change information.												
	Establish a IEEE 802.11ax channel simulator, transmitting images from aircsim to CV model.												
	Refine the channel simulator to send steering data back to theAircsim. Integrate the whole project.												
	Results' analysis and evaluation												
Version 2	Develop GUI for the final project. Prepare detailed reports.												

Figure 1: Gantt chart

A shared space was essential for accumulating all new developments and keeping files updated. To facilitate this, a common GitHub repository has been created. This repository is used to track coding developments, manage PowerPoint presentations, and maintain an Excel sheet for logging all work hours.

Utilizing GitHub ensures effective collaboration, version control, and access to the latest versions of all project materials.

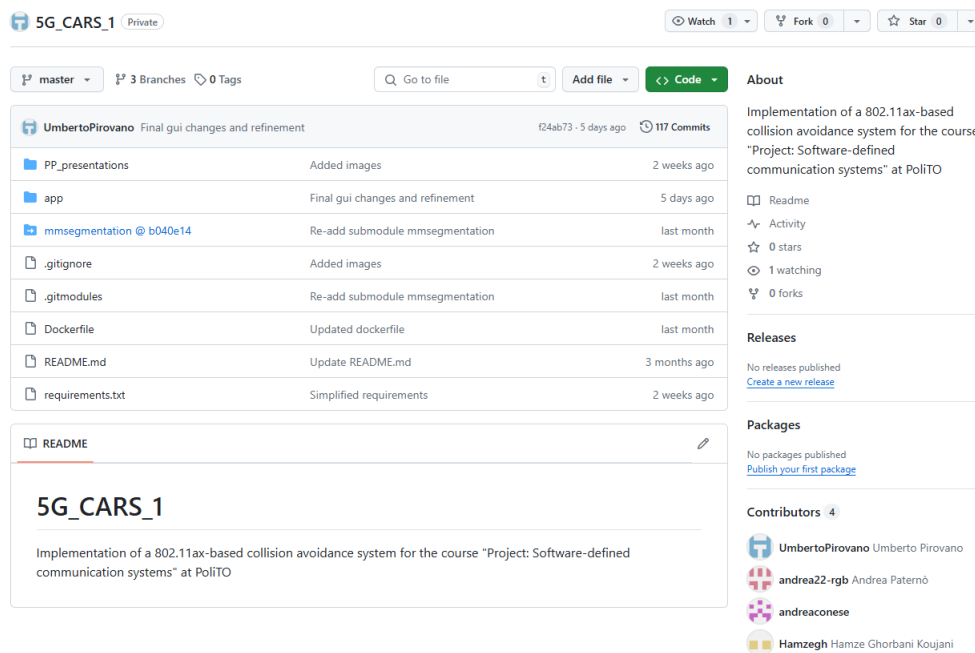


Figure 2: GitHub repository

## 3 Version 0

The initial phase of the project, designated as V0, focused on establishing a robust foundation upon which subsequent development could be built. This phase encompassed a series of crucial tasks, including the configuration of the environment, the integration of the AirSim simulation platform, the selection of a suitable computer vision model, and a thorough exploration of AirSim’s capabilities. The V0 phase, while primarily exploratory in nature, laid the groundwork for the ambitious goals set for V1, encompassing path logic development, control signal generation, wireless channel simulation, and comprehensive system integration.

### 3.1 V0 achievements

#### 1. Environment Setup:

- **Windows/Linux Compatibility:** The project environment was successfully configured to seamlessly operate on both Windows and Linux operating systems, maximizing flexibility and accessibility for developers.
- **Python and Conda Integration:** Python, the project’s primary language, was used in conjunction with the Conda package and environment management system, simplifying dependency management and ensuring a consistent development environment across platforms.
- **Version control:** The code and all the other files related to the project have been made available in a GitHub repository, making development easily accessible to all members of the team. This allowed also an unmatched flexibility in terms of keeping track of changes, and, when needed, reverting them. Something that wouldn’t have been possible with a manual exchange of files.
- **Project organization:** Since the very beginning, the code has also been organized following the most general and simple guidelines of object-oriented programming. Leveraging the tools directly provided by Python, all the main components of the project have been divided in different classes, making them extremely easier to maintain, debug and upgrade. This allowed also each member to focus on different aspects of the project at the same time.

#### 2. AirSim Setup:

- **Scenario Selection and API Installation:** A suitable scenario within the AirSim simulation environment was chosen among the ones available, and the necessary APIs were installed to enable interaction between the simulation and the project codebase.
- **Vehicle Control and Image Capture:** Basic control mechanisms were implemented to manipulate the movement of a virtual car within the AirSim environment. The ability to capture images from the car’s perspective was established, providing the raw data for subsequent computer vision tasks.

#### 3. Computer Vision Model Selection:

- **Research and Evaluation:** A comprehensive research effort was undertaken to identify potential computer vision models, tasks and datasets suitable for the project’s objective.
- **Comparative Analysis:** Various models were assessed based on their performance in object detection and segmentation tasks. Key metrics included execution time, accuracy, and adaptability to the project’s specific requirements.

### 3.2 CV Model Selection

Different models were evaluated based on their performance in object detection and segmentation tasks. Some of these models are presented in the table below.

CV Model	Task	Dataset	Execution Time (seconds)*
YoloV8-m	Object detection	COCO	0.012
MMdetection Faster R-CNN	Object detection	Cityscapes	0.763
MMsegmentation DeepLabV3+	Instance segmentation	Cityscapes	1.807

Table 1: Execution times for different CV models on various tasks and datasets.

\*Note: Execution times are based on processing a single image and may vary depending on hardware configuration (in this case a Nvidia RTX 2060 was used).

Following the initial comparative analysis, the decision was made to pivot towards the MMsegmentation DeepLabV3+ model for the V0 phase. Several factors contributed to this decision:

1. **Cityscapes Pre-Trained Model:** The selected DeepLabV3+ model is trained on the Cityscapes dataset, which closely resembled the urban environments within the AirSim simulation. This pre-training gave the model a head start in understanding and segmenting relevant objects like roads, sidewalks, vehicles, and pedestrians.
2. **Instance Segmentation Capability:** The model’s ability to perform instance segmentation, despite being more computationally expensive, proved to be a significant advantage over simple object detection. By distinguishing individual instances of objects within a scene, the path logic algorithm could more effectively reason about the environment, leading to improved navigation decisions for the virtual car.
3. **AirSim Compatibility:** While not explicitly trained on AirSim data, DeepLabV3+ demonstrated a remarkable ability to generalize to the AirSim environment. This adaptability was crucial for ensuring the model’s effectiveness in processing images captured from the virtual car’s perspective.

#### AirSim Documentation and Sensor Configuration

In parallel with model selection and refinement, a substantial effort was dedicated to understanding and leveraging the capabilities of the AirSim simulation environment. Extensive reading of AirSim’s documentation was undertaken to identify relevant sensors and settings that could enhance the project’s data collection and analysis processes.

The V0 phase, while primarily focused on foundational setup and exploration, laid the groundwork for the ambitious goals set for V1. By successfully establishing a development environment, integrating AirSim, selecting and refining a suitable computer vision model, and delving into the intricacies of AirSim’s capabilities, the project team is now well-equipped to tackle the challenges of distance estimation, control signal generation, wireless channel simulation, and comprehensive system integration. The insights gained and the infrastructure established during V0 will undoubtedly serve as a springboard for the advancements anticipated in the subsequent phases of development.



## 4 Version 1

While the V0 of the project aimed to setting up the work environment and choose the Computer vision model, the V1 focused on implementing an "abstract" of the transmitting channel based on the 802.11ax protocol and the path logic in order to give the command to the car. Key milestones achieved in V1 included:

### 1. Wireless Channel Simulator Setup:

- A Python-based wireless channel simulator was implemented to simulate a basic and elementary channel taking into account a simplified model of losses.
- Channel models and modulation schemes were configured to align with the IEEE 802.11ax standard.

### 2. Path Logic Algorithm Development:

- Obstacle detection capabilities were integrated into the system by defining a region of interest (ROI), enabling the virtual car to identify and react to obstacles in its path.
- A decision-making framework was developed to guide the car's navigation based on the perceived environment and the desired trajectory.
- Real-time adaptation mechanisms were incorporated to ensure the car's responsiveness to dynamic changes in the environment.

### 3. System Assessment and Optimization:

- The region of interest (ROI) within captured images was refined to focus computational resources on relevant areas.
- Performance optimization techniques were applied to reduce round-trip time (RTT) and enhance the system's overall responsiveness.
- Simulation and decision-making parameters were tuned to achieve optimal performance in a variety of scenarios.

## 4.1 Wireless Channel Simulator based on IEEE 802.11ax

In order to simulate the abstract wireless communication environment between the AirSim simulation and the computer vision model, a custom Python-based channel simulator was developed adhering to the IEEE 802.11ax standard. This simulator incorporates critical real-world parameters like path loss, shadowing, and noise to provide a simplified representation of signal propagation and attenuation.

The simulator utilizes the following key components:

**Free Space Path Loss (FSPL) Calculation:** FSPL is calculated based on the distance between the transmitter and receiver. This calculation accounts for the natural attenuation of the signal as it propagates through space. The distance has been computed within the AirSim's coordinate system using the euclidean distance between the car coordinates and assuming that the access point to which the car is connected with is placed in the origin of the axis (0,0,0).

$$A_{f_{psl}} = 20 \cdot \log_2 d + 20 \cdot \log_2 f + 32.44$$

The constant is obtained by using as unit of measurement meter for the distance and GHz for the frequency.

**Shadowing and Fading Effects:** Random shadowing and fading components are introduced to model the unpredictable variations in signal strength caused by obstacles, reflections, and multipath propagation.

$$X \sim N(0, 1 - 4 \text{ dB})$$

**Signal-to-Noise Ratio (SNR) Calculation:** SNR is computed by considering the transmitted power, noise power, FSPL, and shadowing/fading effects. This metric quantifies the quality of the received signal relative to the background noise.

$$SNR[dB] = P_{tx} - P_n - A_{fspl} - A_{sh}$$

**Channel Capacity and Bitrate Estimation:** Once the SNR is computed, in order to obtain the capacity limit the Shannon theorem was used, and by taking into account the code rate, that for our protocol is set to be  $R_c = \frac{5}{6}$ , and an efficiency factor  $\mu = 0.8 - 1$  that takes into account further losses, the bit rate is obtained.

$$C = B \cdot \log_2(1 + SNR)$$

$$R_b = C \cdot R_c \cdot \mu$$

Then by using the size of the image that needs to be sent and by mean of the bitrate the latency to transmit the image is computed:

$$T_x = \frac{File_{size}}{R_b}$$

**Bandwidth Usage and Transmission Time Calculation:** The simulator assesses the utilization of the available bandwidth and calculates the time required to transmit a given image over the simulated channel.

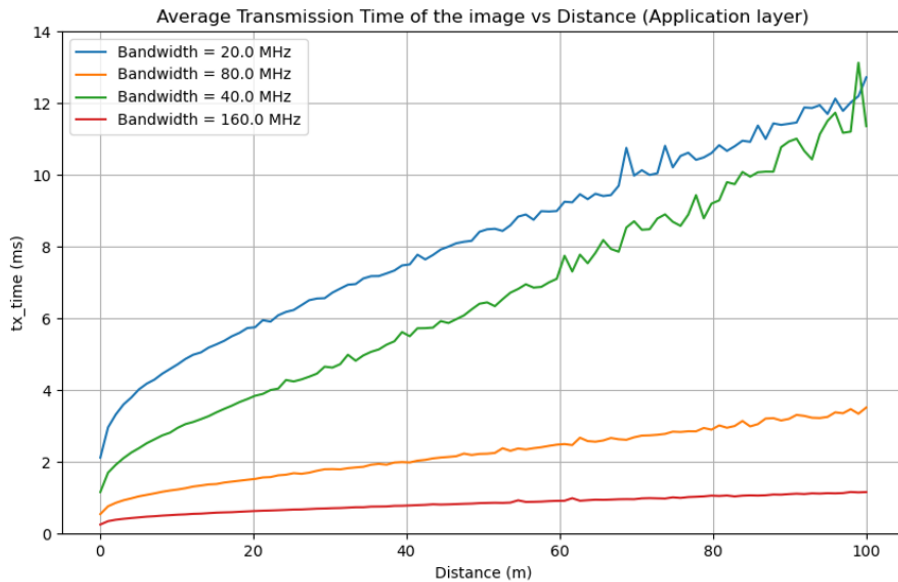


Figure 3: Average transmission time of the image vs distance (application layer) with shadowing component

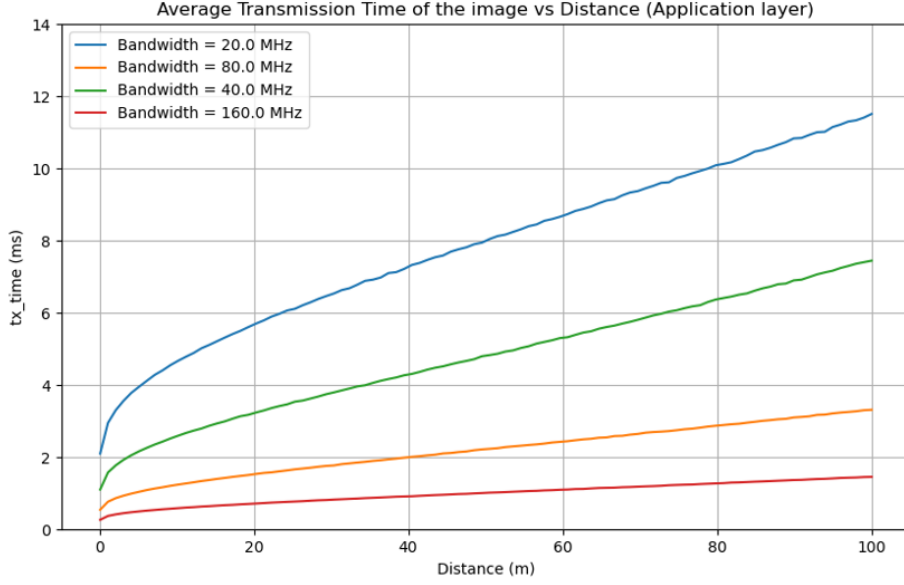


Figure 4: Average transmission time of the image vs distance (application layer) no shadowing component

The included plots illustrate the relationship between average transmission time and distance for different available bandwidths. As expected, higher bandwidths result in shorter transmission times, especially at shorter distances. The plot also reveals the impact of distance on transmission time, with longer distances leading to increased transmission times due to higher path loss and lower SNR. All the values have been assumed taking into account the real values defined in the literature and tuned in order to match the condition in which the simulator should work.

## 4.2 Path Logic: Obstacle Detection and Decision Making

The V1 phase saw significant progress in the development of the path logic algorithm, responsible for controlling the virtual car's speed through the AirSim environment. This algorithm leverages the instance segmentation capabilities of the DeepLabV3+ model to identify and classify objects in the car's path, enabling real-time decision-making and adaptation.

The path logic algorithm comprises two key components:

1. **Obstacle Detection:** The RoI class in the provided Python code implements the obstacle detection functionality. It divides the image into multiple regions of interest (ROIs) based on predefined ratios and steering angle. The class then analyzes the segmented mask within each ROI to determine the presence and type of objects, such as cars, pedestrians, traffic signs, and road markings. The percentage of area occupied by each object type within each ROI is calculated, providing crucial information for decision-making.
2. **Decision Making:** The decision-making component of the path logic algorithm translates the obstacle detection data into actionable commands for the virtual car. The algorithm evaluates the occupancy of each ROI sub-area and applies slowdown factors based on pre-defined thresholds and object priorities. Hence the throttle is reduced while the car is approaching the detected object. To avoid abrupt stops, the algorithm gradually reduces the throttle based on the proximity and type of obstacles,

ensuring a smoother and safer driving experience. A black list of labels was defined comprehending all those objects in the considered Airsim's environment that can pose a threat.

$$Throttle_{new} = Throttle_{target} - \sum_{N=1}^{N_{subregions}} \left( W_N \cdot \frac{OccupiedArea_N}{100} \cdot S_{object} \right)$$

where  $W_N$  is the weight associated to each sub-area while  $S_{object}$  is the weight associated to the objects in order to increase or decrease their weight on the decision.

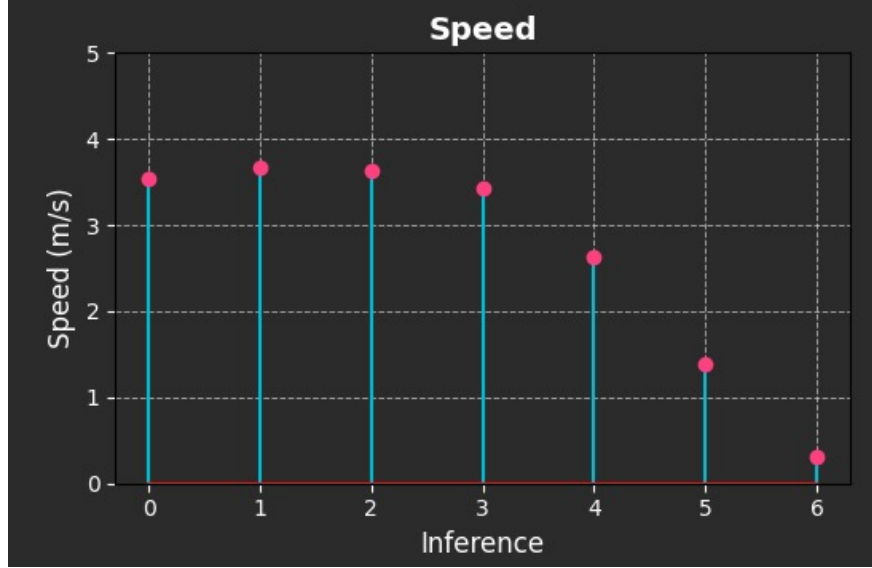


Figure 5: Speed trend

The implemented code optimizes the ROI calculation to consider the steering angle, allowing for a dynamic adjustment of the focus area based on the car's direction.

In addition, the algorithm effectively classifies objects within each ROI using the segmented mask generated by the DeepLabV3+ model. Different slowdown factors are applied based on the type of detected object, prioritizing safety-critical obstacles like vehicles.

Finally, the throttle control is dynamically adjusted based on the detected obstacles and their relative positions, ensuring smooth and safe brakes.

In conclusion, the V1 phase successfully developed a sophisticated path logic algorithm that empowered the virtual car with the ability to perceive its environment, make intelligent decisions, and avoid collisions safely. This achievement marked a significant milestone in the project.

### 4.3 Performance Analysis

The system's performance was a primary concern during the V1 phase. Several key aspects were analyzed to assess the system's efficiency and identify potential bottlenecks.

#### Image Capture and Transmission:

Capturing images in PNG format without any compression from AirSim was found to take approximately 200 ms. The transmission time for uploading the captured images was relatively low, thanks to the optimized wireless channel simulator. However, the uncompressed PNG files, with an average size of 1000 KB, posed a potential bottleneck in the transmission

process. To address this, JPEG compression was implemented, which significantly reduced the image file size to 100 KB. This compression not only reduced transmission time but also led to improvements in other sections of the simulation, including image capture and processing. The total average time for these operations was reduced from approximately 2.235 seconds to 1.889 seconds, highlighting the significant impact of image compression on overall system performance.

```
Average time for capture: 0.2135082483291626
Average time for tx: 0.010479903221130371
Average time for inference: 2.0111788272857667
Total average time: 2.2351669788360597
```

Figure 6: Png, no compression

```
Average time for capture: 0.06331236021859306
Average time for tx: 0.0038505452019827707
Average time for inference: 1.8218083722250802
Total average time: 1.888971277645656
```

Figure 7: Jpeg, quality: 80%

### Segmentation Model Inference:

The most computationally intensive task was the segmentation model inference, which could take up to 2000 ms per image due to hardware limitations. This bottleneck underscored the need for further optimization in future phases, potentially through hardware upgrades or model compression techniques. The inference time directly impacted the system’s overall responsiveness and real-time capabilities.

### Impact of Model Complexity:

Four different computer vision models with varying complexity (super\_light, light, medium, heavy) were integrated and tested within the system. As expected, the heavier models, while potentially offering higher accuracy, demanded significantly more memory and exhibited lower inference speeds compared to the lighter models. This trade-off between accuracy and speed was a crucial factor in model selection and optimization. The "light" model was chosen for subsequent analysis due to its balanced performance in terms of accuracy and speed, making it suitable for real-time applications.

TAG:	super_light	light	medium	heavy
MEMORY REQUIRED (GB)	2.2	6.35	7.5	12.5
INF. TIME (FPS) NVIDIA RTX 2060	0.8	0.65	0.55	0.5
INF. TIME (FPS) NVIDIA V100	14.27	7.87	3.94	1.15

Table 2: Performance analysis of different CV models.

**Hardware Limitations:**

The performance analysis revealed that the available hardware, namely the NVIDIA RTX 2060, was a limiting factor in achieving satisfying real-time performance, especially for the heavier segmentation models. The analysis suggests that a more powerful GPU, such as the NVIDIA V100, could significantly improve inference speeds and overall system responsiveness.

In conclusion, the performance analysis conducted during the V1 phase provided valuable insights into the system's bottlenecks and areas for improvement. The findings highlighted the need for further optimization in model selection, hardware upgrades, and algorithm refinement to achieve the desired real-time performance and responsiveness in future iterations.

## 4.4 V1 Conclusion

In conclusion, the V1 phase successfully addressed critical aspects of the project, including the development of an abstract wireless channel simulator, the implementation of a robust path logic algorithm, and the comprehensive analysis and refining of system performance. The insights gained from the performance analysis, particularly regarding the impact of model complexity and hardware limitations, will be invaluable in guiding future development efforts. The V1 phase has not only demonstrated the feasibility of the proposed system but also paved the way for real-world implementation and testing.

Finalizing the V1 milestones allows us to transition to the V2 phase, where the focus will shift towards refining the simulator and developing a user-friendly graphical user interface (GUI). The GUI will enable seamless interaction between the simulator and AirSim, allowing users to easily configure simulation parameters and visualize the system's performance. This user-centric approach will enhance the accessibility and usability of the system.

## 5 Version 2

The most recent updates have been incorporated into the project's final version, along with a graphical user interface (GUI) whose functionality will be discussed thereafter.

### 5.1 GUI implementation

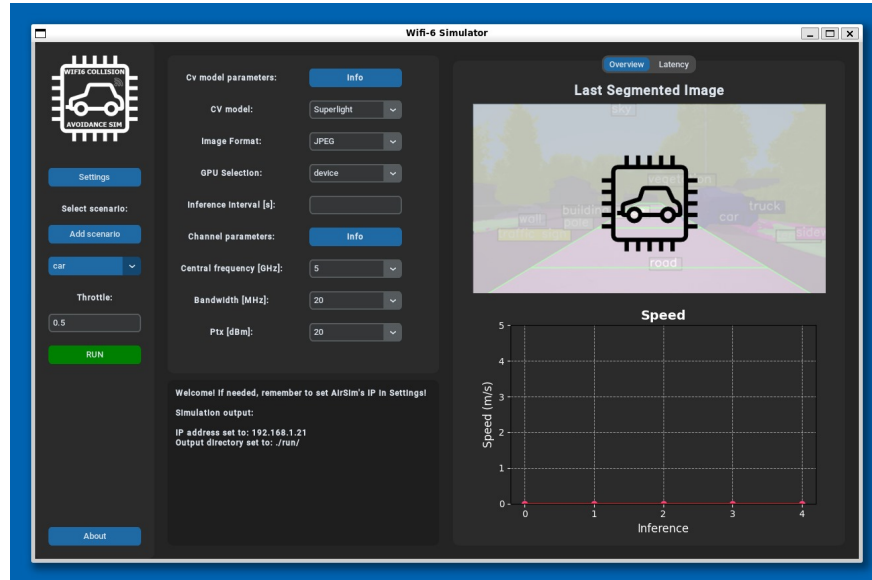


Figure 8: GUI

The following image illustrates how the user can select from a variety of options to launch various kinds of simulations. Let's move step by step so that you can read and comprehend how to utilize the software more easily. To begin the simulation, the initial parameters must first be set up.

### 5.2 Settings

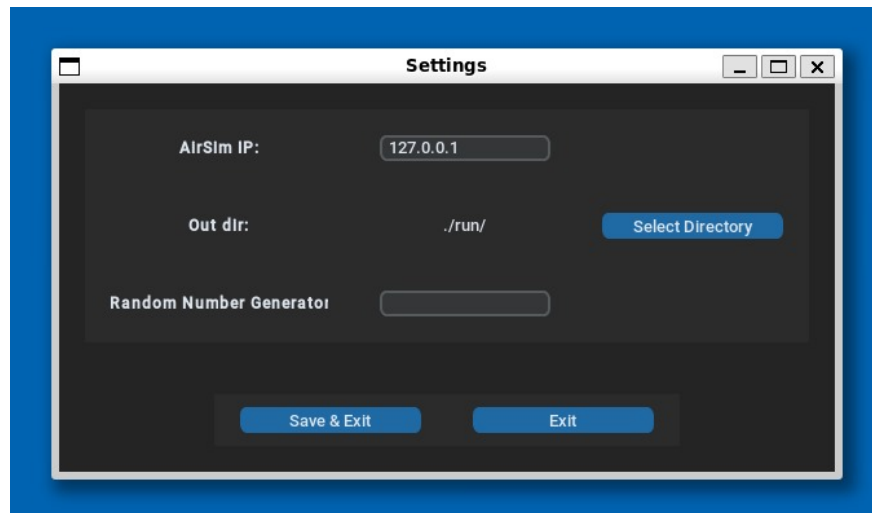


Figure 9: Settings

To run the simulation on a different device, the **IP address** can be entered; otherwise, the local host address will be used. Next, the **folder** in which to save the simulation's progress data must be chosen.

### 5.3 Scenario editor

Next, the preferred scenario can be selected for the simulation.

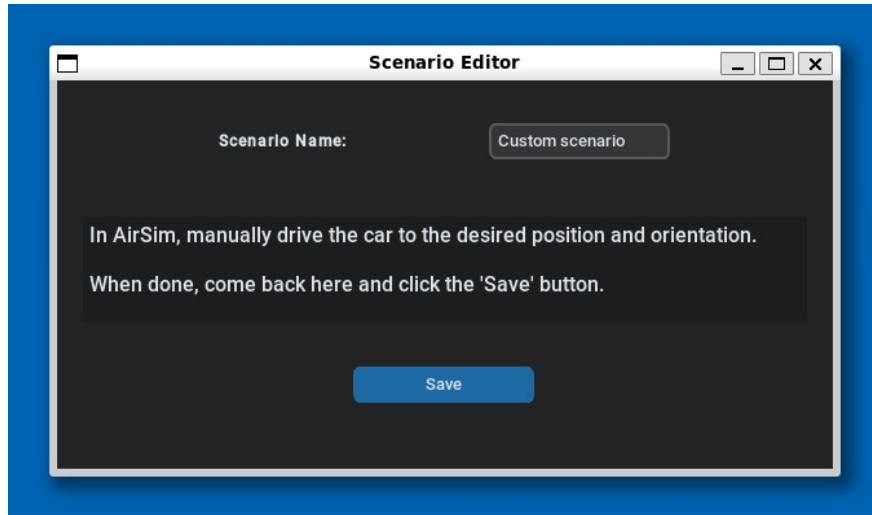
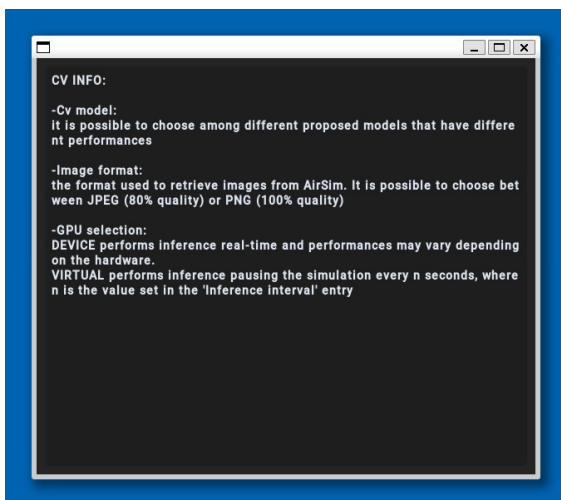


Figure 10: Scenario editor

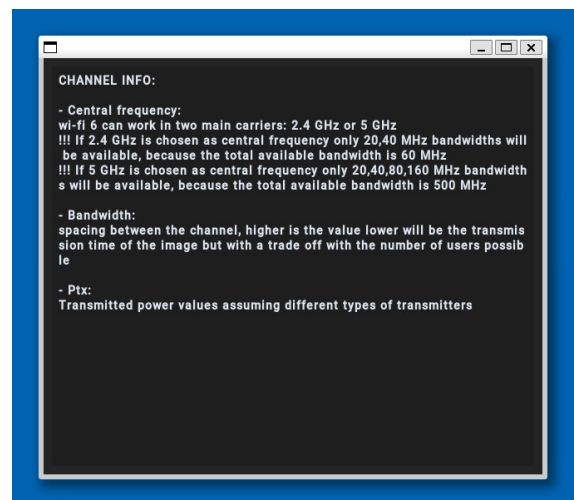
The user can add additional scenarios at any time, although there are two basic scenarios available: **car** and **fence**. To complete this process, the vehicle must be manually moved to Airsim, and the beginning point of the simulation is saved at these coordinates. After that, an initial value for the car's throttle can be chosen.

### 5.4 Simulation parameters

Next, the section that deals with the **channel's** and the **CV model's parameters** is addressed. After clicking the **info** button, information and assistance windows will open. It is important to follow the instructions in these windows in order to proceed correctly.



(a) CV model info



(b) Channel info

Figure 11: Simulation parameters



## 5.5 Graphical results

After making all the necessary setups, the simulation can be started by clicking the **RUN** button and examine the outcomes later. In the area to the right of the GUI two tabs are available to consult the results: one for a general **overview** and another specifically for **latencies**.

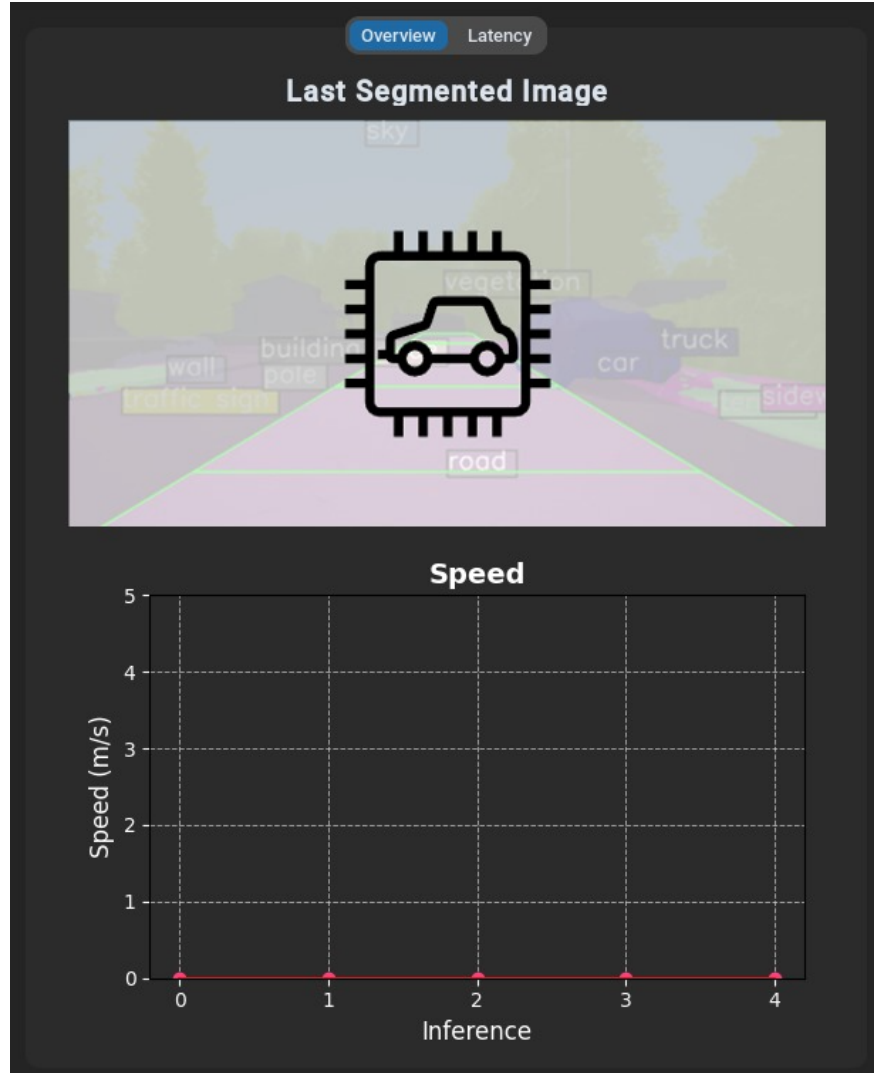
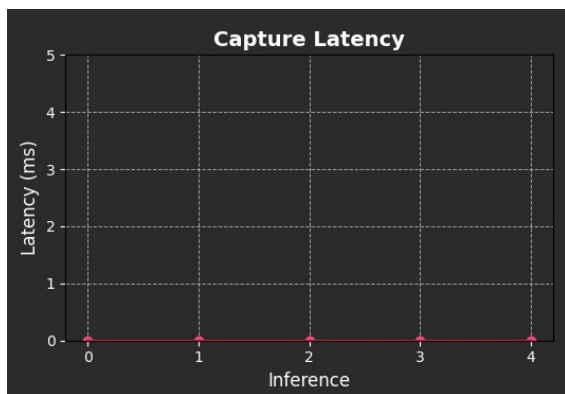


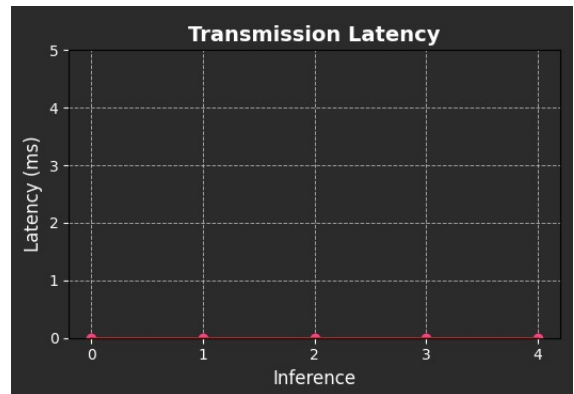
Figure 12: Overview: Last segmented image and speed trend

In the overview tab, in chronological sequence, the first image displays to the user the most recent image on which image segmentation was performed. Beneath that, a graph illustrates the vehicle's speed whose trend varies based on the real distance between it and the obstacle in question. However, if the latency option is chosen, four basic graphs will be provided:

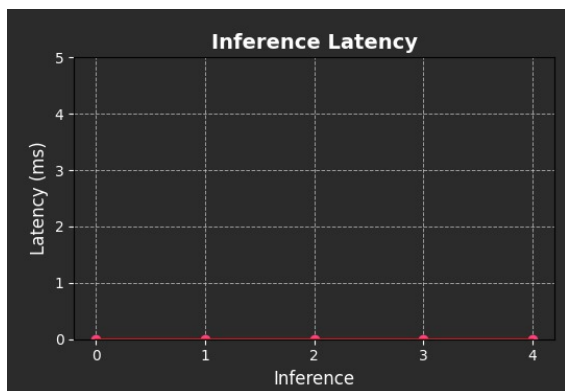
- capture latency
- inference latency
- transmission latency
- overall latency



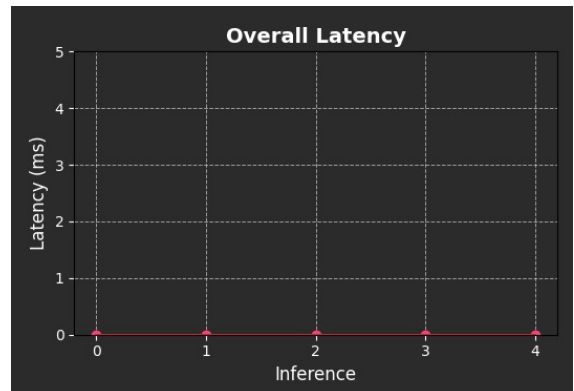
(a) Capture latency



(b) Transmission latency



(c) Inference latency



(d) Overall latency

Figure 13: Latency

## 5.6 Output window

Below the section dedicated to the general parameters to be set, there is an output window that gives the user a general summary of what is happening during the simulation process.

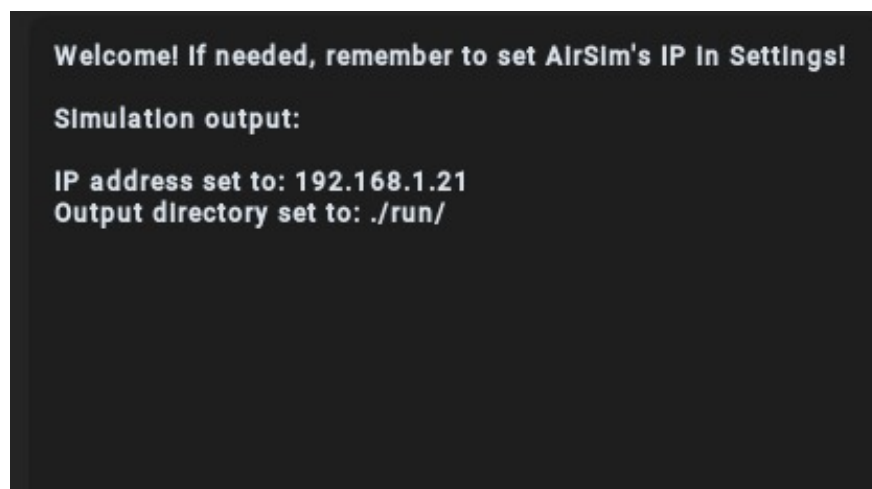
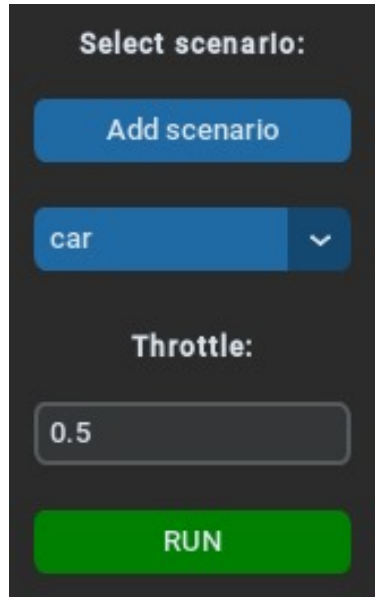


Figure 14: Output window

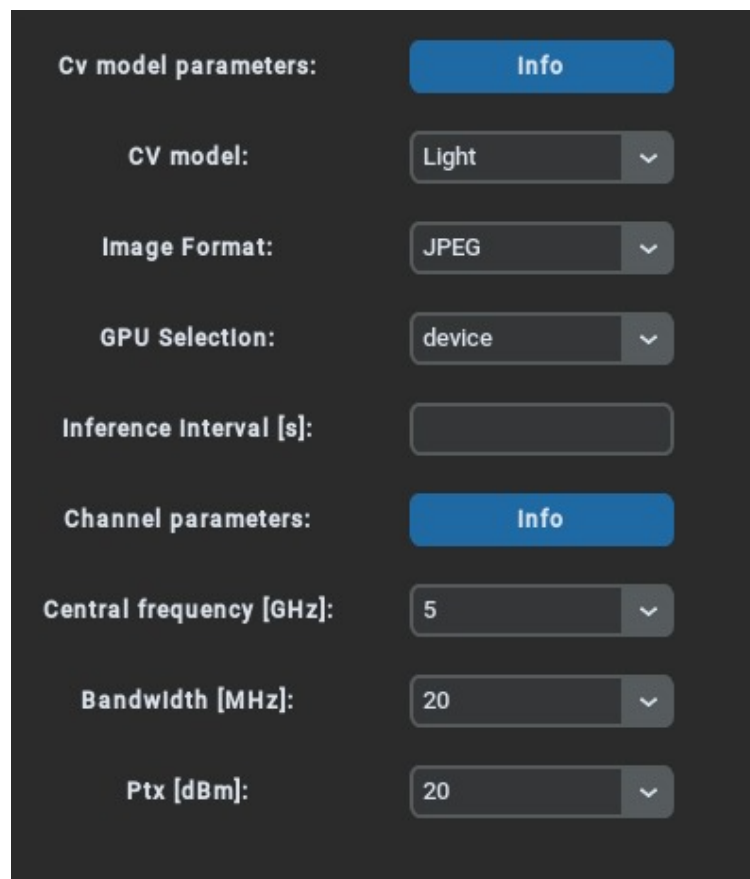
## 5.7 Simulation example

The next section is devoted to a simulated example in order to elucidate the previously discussed points and provide tangible outcomes for the incredible job undertaken to arrive at this point. Below are the images that encapsulate all the choices made before starting the collision avoidance simulation with another car.



The image shows a dark-themed control panel. At the top, it says "Select scenario:". Below this is a blue button labeled "Add scenario". Underneath is a dropdown menu currently showing "car" with a downward arrow. Further down, it says "Throttle:" followed by a text input field containing the value "0.5". At the bottom is a large green button labeled "RUN".

Figure 15: Scenario and throttle



The image shows a dark-themed configuration panel. It is divided into two main sections. The top section is titled "Cv model parameters:" and includes an "Info" button. Below this title are four settings: "CV model:" with a dropdown set to "Light", "Image Format:" with a dropdown set to "JPEG", "GPU Selection:" with a dropdown set to "device", and "Inference Interval [s]:" with an empty text input field. The bottom section is titled "Channel parameters:" and also includes an "Info" button. Below this title are three settings: "Central frequency [GHz]:" with a dropdown set to "5", "Bandwidth [MHz]:" with a dropdown set to "20", and "Ptx [dBm]:" with a dropdown set to "20".

Figure 16: Simulation example parameters

```

Welcome! If needed, remember to set AirSim's IP in Settings!

Simulation output:

IP address set to: 192.168.1.21
Output directory set to: ./run/
Starting simulation...
Selected CV model: Light
Selected Image format: JPEG
Selected GPU: device
Selected channel parameters: [20000000.0, 5.0, 20]
Scenario 1 completed: No collision detected.
Simulation completed.

```

Figure 17: Output window example

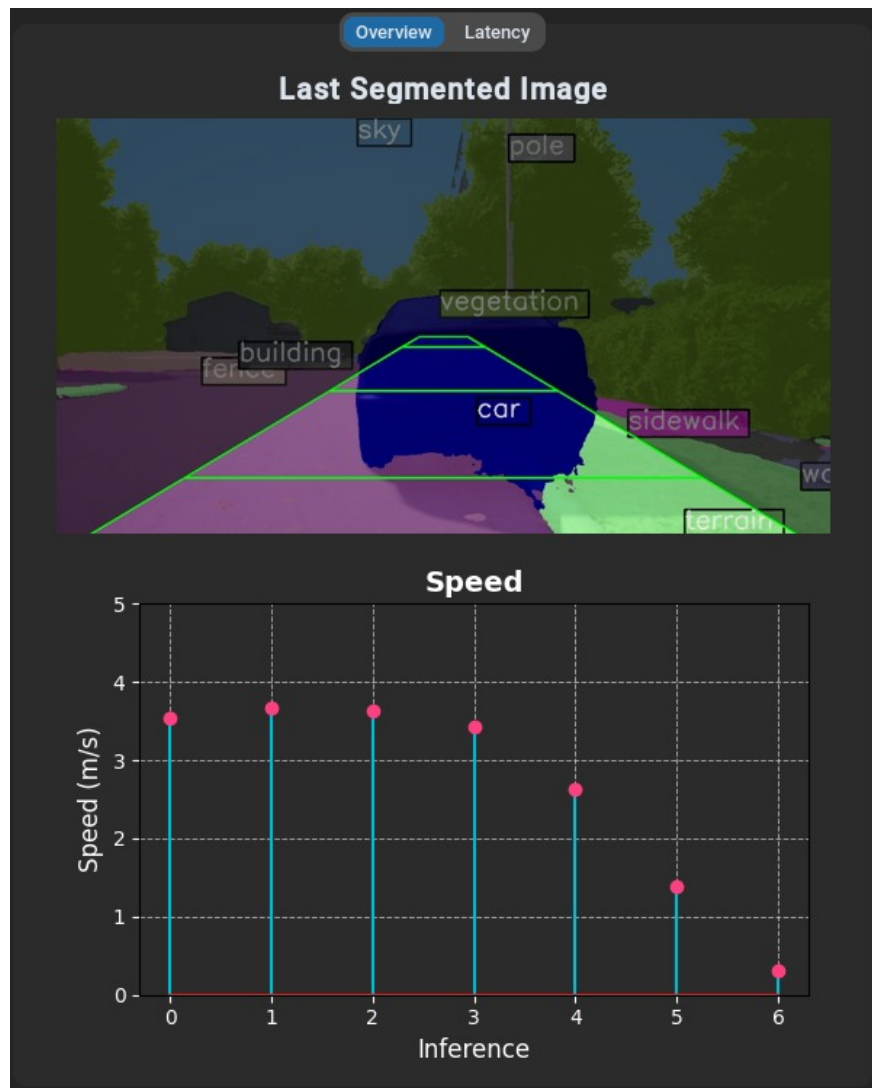
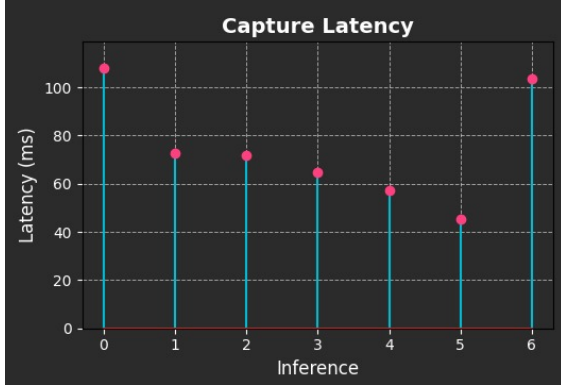


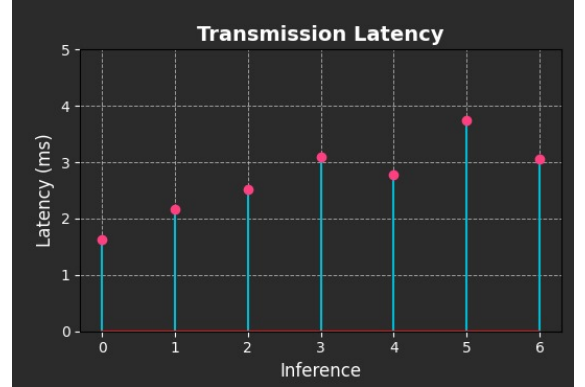
Figure 18: Last segmented image and speed trend example

By analyzing the previous image, you can distinguish the last segmented image and the graph of speed versus distance. Obviously, the first image represents the moment when the

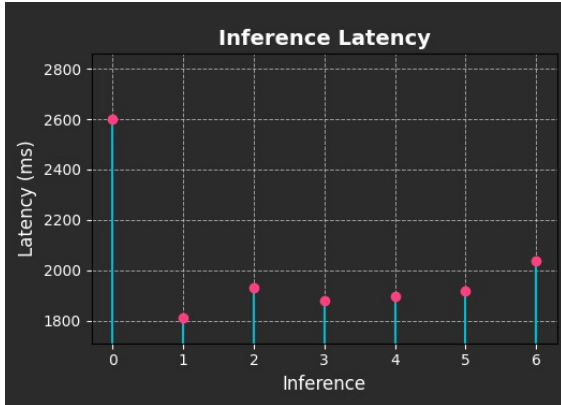
car stops to avoid collision with the other car. The plot makes it rather evident how the car slows down as it gets closer to the obstacle in front of it. To better clarify this aspect, it is possible to note that each plot has the x-axis called "inference", this concept is directly related to the distance between the car and the obstacle considered. More precisely, the number of inferences changes depending on the initial distance between the car and the obstacle in the selected scenario: a wider distance will result in a longer simulation time and therefore more inferences will be performed. Be aware though that the amount of inferences is affected also by other factors and, for instance, choosing a more efficient model or a more capable wireless channel would also result in a higher amount.



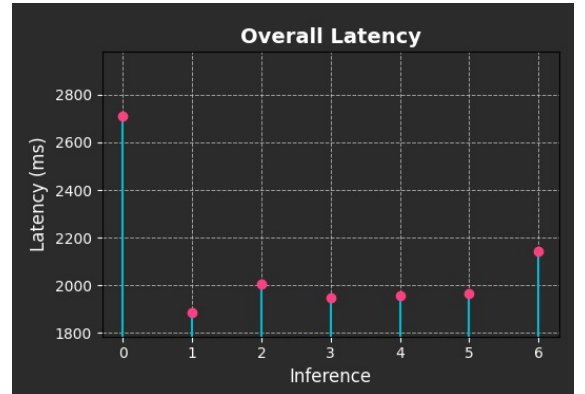
(a) Capture latency



(b) Transmission latency



(c) Inference latency



(d) Overall latency

Figure 19: Latency

From the four latency graphs, it can be seen that the determining factor in overall latency is the time it takes to make each individual inference, so this process is what we pay for in terms of time.

## 6 Final comments and suggestions

The final project successfully achieved its objectives, demonstrating the feasibility and potential of the integrated system. Through effective collaboration and organization, the team accomplished the development of this simulator within a tight three-month timeframe, concluding at the end of June. This efficient workflow allowed team members, who are also students, to dedicate their time to studying for exams in July.

### 6.1 Extra Features

In addition to the core milestones, the team implemented extra features that enhanced the simulator’s functionality and usability:

1. **Offline Segmentation:** This feature pauses the AirSim simulation during inference to focus computational resources on the segmentation process. This is particularly beneficial when dealing with hardware limitations, as it prevents the simulation suffering from being held back by the computationally intensive segmentation task.
2. **Custom Scenarios:** Users can now manually drive the car within the simulation environment to a desired position and save it as a named scenario. This feature enables the creation of personalized testing environments and facilitates the exploration of specific driving conditions or challenges.

These additional features, along with the core functionalities developed in V1, have laid a solid foundation for further advancements in the project.

### 6.2 Suggestions

The V1 phase successfully achieved its objectives, demonstrating the feasibility and potential of the integrated system. However, several areas for improvement and further exploration have been identified:

1. **Hardware Acceleration:** The performance analysis highlighted the limitations of the current hardware, particularly in the context of real-time image segmentation. Investing in more powerful GPUs or exploring hardware acceleration techniques.
2. **Algorithm Refinement:** The path logic algorithm, while functional, could be further refined to improve decision-making in complex scenarios. Incorporating machine learning techniques or exploring alternative decision-making frameworks could lead to more sophisticated and adaptable navigation strategies.
3. **Real-World Testing:** The current system has been primarily evaluated in a simulated environment. Transitioning to real-world testing with a physical vehicle would be a crucial next step. This would expose the system to a wider range of environmental conditions and challenges, providing valuable insights for further development.
4. **Sensor Fusion:** The current system relies primarily on visual input from a single camera. Integrating additional sensors like LiDAR or radar could enhance the system’s perception capabilities, particularly in adverse weather conditions or low-light environments.

By addressing these areas in future development phases, the project has the potential to significantly advance the state-of-the-art simulation in autonomous vehicle technology.

## 7 User guide

Various options have been considered for the deployment of this application, ranging from Docker to producing an executable. Unfortunately, python, the extensive use of external libraries and the presence of a user interface make this task quite challenging. For this reason, as of now, the suggested option is to use the simulator cloning the repository and manually installing the needed dependencies. Anyway, a Docker image is available, and it is an option to avoid the installation of all the dependencies.

### 7.1 Docker

If you want to just run the simulator avoiding the hassle of installing (almost) anything, it is possible to do so using the provided Docker image.

Since there is a GUI, how to run the image depends on the operative system you are working on.

Ubuntu and WSL2 already come with a X server installed (necessary to visualize the interface), hence it is enough to run the command:

```
1 docker run --rm -it --network=host --name simulator -e DISPLAY=$DISPLAY
2 -v /tmp/.X11-unix:/tmp/.X11-unix:ro collision_sim:latest
```

On Windows, it is a bit more tricky, and you must install manually an X server application (e.g.: Xming) and set it up correctly to work.

### 7.2 Installation guide

This is the traditional approach that allows you also to modify the code and develop it further.

#### 7.2.1 Supported Operative System

The project has been fully developed in Linux, hence this is the supported os. Being the project heavily hardware dependent, the suggested solution is to use Windows Subsystems for Linux (WSL2) with Ubuntu 22.04. This will allow you to virtualize a Linux system on your Windows machine maintaining the access to all the precious resources available like GPU and RAM, something not possible with traditional virtual machines.

On Windows 11, to install WSL2 and Ubuntu, open the Microsoft Store and then look for 'Ubuntu 22.04.3 LTS' and install it. Once done, by starting this application, a Linux shell will appear. All the steps that will be presented in the next sections are intended to be performed within this shell. The only exception is running AirSim, which can be done on Windows.

#### 7.2.2 AirSim

AirSim provides a series of pre-built binaries easy to execute. These are more than enough to use the simulator, hence you can download the one you prefer from their official GitHub.

It is suggested to use the small urban scenario "**AirSimNH**" since the development of the

simulator has been based on it. Note also that the predefined simulation scenarios will work correctly only within this AirSim scenario.

### 7.2.3 Cloning the repository

Create a workspace directory and clone the repository inside it using the following command:

```
1 git clone --recursive https://github.com/UmbertoPirovano/5G-CARS1.git
```

Be sure to include "--recursive" in the command, otherwise the mmsegmentation submodule will not be downloaded.

### 7.2.4 Dependencies

Once inside the repository respective folder, it is necessary to install all the dependencies necessary for the project. For this purpose it is suggested to create a virtual environment with VENV and work inside it. The same purpose could be achieved using Conda, but in that case some graphic issues could arise when using the GUI.

```
1 python -m venv myenv
```

Remember to activate the virtual environment every time you work on this project:

```
1 source myenv/bin/activate
```

At this point you can proceed installing the dependencies with "pip".

- General needed dependencies:

```
1 customtkinter==5.2.2
2 ftfy==6.2.0
3 numpy==2.0.0
4 pillow==10.3.0
5 regex==2024.5.15
6 requests==2.32.3
7 scipy==1.13.1
8 matplotlib==3.9.0
```

- AirSim APIs:

```
1 pip install msgpack-rpc-python
2 pip install airsim
```

- Pytorch:

Install Pytorch following the official guide e.g:

On NVIDIA GPU platforms:

```
1 pip3 install torch torchvision torchaudio
```

On CPU-only platforms:

```
1 pip3 install torch torchvision torchaudio --index-url
2 https://download.pytorch.org/whl/cpu
```

- MMCV and MMSEGMENTATION:

Install mmdcv and mmsegmentation following the official guide e.g:



```
1 pip install -U openmim
2 mim install mmengine
3 mim install "mimcv>=2.0.0"
4 pip install mmsegmentation
```

Installing mmcv can often be challenging, so it's important to approach the process carefully. If the installation takes a while, it usually indicates that the package is being built correctly and installed properly.

During the installation of these dependencies some errors might be raised. If it is so, look carefully at the error message. In many cases it could be that the error is caused by another dependency missing, and it can be fixed installing it.

Once you have completed all the steps above, everything should be set to run the simulator.

### 7.3 Run the simulator

Move inside the project folder and activate the virtual environment:

```
1 source myenv/bin/activate
```

Run the script:

```
1 python ./app/GuiApp.py
```

At this point the GUI should appear. If you haven't done so yet, you can launch the AirSim executable, and then run the simulation.

Note: if AirSim is running on another machine, or on the same machine but another OS, you need to specify the IP address of the host machine in the simulator's settings (this step is mandatory if you are using the suggested WSL2 setup while running AirSim on Windows).