



# Driver Identification and Detection of Drowsiness while Driving

Sonia Díaz-Santos <sup>1,\*</sup> , Óscar Cigala-Álvarez <sup>1</sup>, Ester Gonzalez-Sosa <sup>2</sup>, Pino Caballero-Gil <sup>1</sup> and Cándido Caballero-Gil <sup>1</sup> 

<sup>1</sup> Department of Computer Engineering and Systems, University of La Laguna, 38271 Tenerife, Spain; ocigalaa@ull.edu.es (Ó.C.-Á.); pcaballe@ull.edu.es (P.C.-G.); ccabgil@ull.edu.es (C.C.-G.)

<sup>2</sup> eXtended Reality Lab, Nokia, 28045 Madrid, Spain; ester.gonzalez@nokia.com

\* Correspondence: sdiazsan@ull.edu.es

**Abstract:** This paper introduces a cutting-edge approach that combines facial recognition and drowsiness detection technologies with Internet of Things capabilities, including 5G/6G connectivity, aimed at bolstering vehicle security and driver safety. The delineated two-phase project is tailored to strengthen security measures and address accidents stemming from driver distraction and fatigue. The initial phase is centered on facial recognition for driver authentication before vehicle initiation. Following successful authentication, the subsequent phase harnesses continuous eye monitoring features, leveraging edge computing for real-time processing to identify signs of drowsiness during the journey. Emphasis is placed on video-based identification and analysis to ensure robust drowsiness detection. Finally, the study highlights the potential of these innovations to revolutionize automotive security and accident prevention within the context of intelligent transport systems.

**Keywords:** facial recognition; drowsiness detection; driver safety; machine learning; video-based identification



**Citation:** Díaz-Santos, S.; Cigala-Álvarez, Ó.; Gonzalez-Sosa, E.; Caballero-Gil, P.; Caballero-Gil, C. Driver Identification and Detection of Drowsiness while Driving. *Appl. Sci.* **2024**, *14*, 2603. <https://doi.org/10.3390/app14062603>

Academic Editor: João M. F. Rodrigues

Received: 5 February 2024

Revised: 11 March 2024

Accepted: 18 March 2024

Published: 20 March 2024



**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Vehicular theft has been a challenge since the automobile's inception in the 19th century, impacting both manufacturers and individual owners. Traditional security measures, ranging from mechanical locks and alarms to electronic immobilizer systems, have demonstrably mitigated such threats. However, the evolving technological landscape necessitates the development of more sophisticated and robust security solutions. The integration of Internet of Things (IoT) capabilities presents a promising avenue for enhancing road safety and vehicle security. In this context, in-vehicle authentication emerges as a critical component for thwarting unauthorized access attempts.

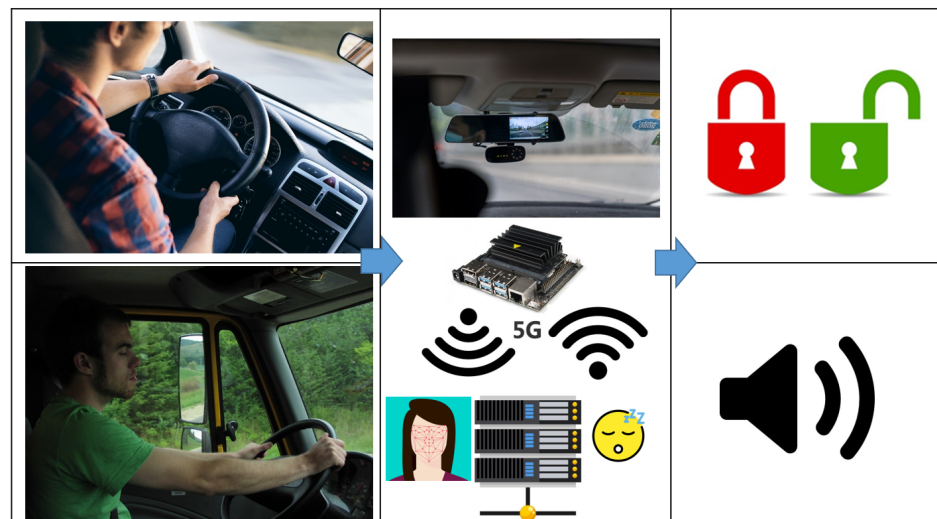
In addition to this, traffic accidents are responsible for approximately 1.3 million deaths worldwide each year, with young people being the most affected group. Distracted driving and fatigue rank among the leading causes of these accidents, emphasizing the necessity of mitigating these factors to reduce the number of victims. Presently, advanced vehicle models incorporate driver drowsiness and attention warning (DDR-AW) [1] as one of their safety technologies, integrating the smart components of intelligent transport systems (ITSs) to prevent accidents caused by driver inattention. ITSs currently play a pivotal role in enhancing road safety, thanks to different components such as real-time traffic data, predictive analytics [2], and smart infrastructure solutions.

In particular, in this work, the combination of innovative technologies with artificial intelligence (AI) techniques is proposed to identify facial signs. This can help, firstly, to identify the driver to provide access to control the vehicle, and secondly, to characterize sleepiness or fatigue during driving through video identification. In this way, it will allow us to leverage the capabilities of edge computing to utilize them in any vehicle.

The article [3] reported by the General Directorate of Traffic in Spain (Dirección General de Tráfico (DGT)) contains a critical aspect that has garnered attention from researchers

and authorities: the prevalence of driver drowsiness as a contributing factor to traffic accidents. In this regard, a detailed examination of the figures provided in the DGT report can shed light on the magnitude of this issue and guide future initiatives to enhance safety on Spanish roads. The report reveals several pertinent figures that underscore the imperative of implementing measures to address traffic accidents associated with fatigue or somnolence. Notably, in 2020, a total of 67 fatal accidents were attributed to this cause, constituting 9% of the concurrent factors contributing to accidents. Furthermore, there were 1458 accidents with victims linked to tiredness or sleepiness, representing 8% of the various factors contributing to accidents. These statistics highlight the critical need for targeted interventions to mitigate the impact of driver fatigue on road safety.

This work has been developed in two fundamental stages (Figure 1). The first phase was designed to carry out facial recognition of the driver when attempting to start the vehicle, rigorously verifying whether the driver possesses the required authorization to operate it. Once this initial stage was completed successfully, the second phase involved the constant monitoring of the driver's characteristic signs of drowsiness throughout the journey, using facial features and the condition of their eyes as reference points, aided by IoT sensors. If a prolonged eye closure is detected, an alarm is activated through the vehicle's speakers to prevent potential accidents.



**Figure 1.** System operation for the two use cases.

This paper is structured as follows. Sections 2 and 3 provide context via a review of previous research and a discussion of different technical aspects, respectively. Subsequently, Section 4 describes the system design, and Section 5 delves into practical implementation details. Finally, Section 6 shows the results and Section 7 summarizes the main findings and their implications.

## 2. Related Work

Innovative strategies for enhancing vehicle security and deterring theft have been a focus of recent research. A notable development in this area can be found within [4], which introduces a system based on facial recognition to safeguard against vehicle theft, effectively employing global systems for mobile communications (GSMs) and Arduino [5] technologies. Additionally, the authors of [6] propose a vehicle tracking system that integrates GSM communication with global positioning system (GPS) technology, enabling real-time monitoring through short message service (SMS). The paper [7] aims to detect driver fatigue by analyzing vehicle operation data, proposing a system using machine learning to distinguish between fatigued and non-fatigued drivers based on driving behavior. The study [8] explores the early detection of driver drowsiness using ensemble machine learning, combining multiple sensors. It proposes a detection system merging physiological

(e.g., heart rate and electromyographic activity) and vehicle sensors (e.g., steering angle and acceleration). These methodologies underscore the importance of integrating advanced technological systems to improve vehicle security, marking a significant stride in the use of novel technologies to prevent car theft and provide both efficient and accessible solutions.

In the scientific literature, numerous diverse approaches addressing road safety, particularly driver behavior, using AI techniques can be found. For instance, the authors of [9] analyzed automated systems that identify improper human driving behavior and proposed a self-adaptation model for trust management based on deep learning, which first identifies safe and unsafe drivers and then classifies the behaviors of safe drivers using deep learning.

The survey in [10] covers different studies that employed at least one camera to observe a driver inside a vehicle. In fact, one of the most difficult issues related to the use of video devices in automotive applications is the need for a high frame rate. In this sense, the authors of [11] proposed a preliminary version of a motion magnification method that addressed this issue.

Moreover, several very recent bibliographic references focus on the application of different AI models based on machine learning (ML) to detect driver drowsiness. One of the first reviews of the literature on driver drowsiness detection based on behavioral measures using ML techniques is [12].

Stacked deep convolutional neural networks (CNNs) were used in [13] to detect faces and extract the eye region from facial images to classify the driver as sleep or non-sleep. The authors of [14] presented a driver drowsiness-detection model based on deep learning techniques for the brain-computer interface. The authors of [15] also suggested the use of a CNN technique as the best machine learning algorithm to detect microsleep and drowsiness. Additionally, the authors of [16] proposed another real-time driver drowsiness-detection system based on the area of eye closure and the use of a CNN [17].

The authors of [18] used a public dataset to propose an architecture that detects driver drowsiness by comprising four deep learning models to extract four different types of features.

The authors of [19] analyzed driver behavior and detected signs of drowsiness using a variety of sensor data, such as cameras, motion sensors, and physiological data, to train deep learning models. A different approach based on the study of emotions through facial images using CNN to extract relevant emotional features is followed in [20]. A modified deep learning architecture proposing specific adaptations in the neural network was used in [21] to analyze and process relevant data, such as driver images and driving patterns. Another approach to deep learning-based driver sleep detection is found in [22], where an efficient approach involving data acquisition and processing, the selection of deep neural network architectures, and performance evaluations of the resulting model is proposed. The methodology employed in [23] is the combination of long-term and long short-term memory CNNs and factored bi-linear features.

None of the aforementioned works, although relevant to the subject matter, fully align with the approach adopted in this paper. The objective of this study is to achieve real-time monitoring of a driver's eye status, coupled with the quickest possible authorization of the individual intending to operate the vehicle. This approach is imperative for several reasons. Firstly, the end-user, represented by the driver, is unlikely to tolerate any delay in facial recognition authorization to commence vehicle operation. Moreover, in emergency situations, waiting for facial recognition authentication is impractical.

Regarding drowsiness detection, transmitting frames can impede system performance and lead to failures. Thus, frames are dispatched as expeditiously as possible and processed at maximum speed on the server. Consequently, the server hosts the drowsiness analysis. A Jetson Nano processor with only 2 GB of memory incurs computation delays when analyzing camera-captured frames, but it is usable in situations such as a loss of communications with the cloud.

Consequently, while the existing literature predominantly discusses identification strategies using ML, this paper's focus lies in implementing a real-time system to ensure

swift analysis and minimize errors. Specifically, this includes mitigating the impact of extended frame transmission intervals, which diminishes the system's efficacy in promptly alerting the driver to potential drowsiness. Additionally, it involves optimizing frame analysis speed to prevent delays in frame capture and avoid analyzing outdated frames.

### 3. Requirements and Technologies

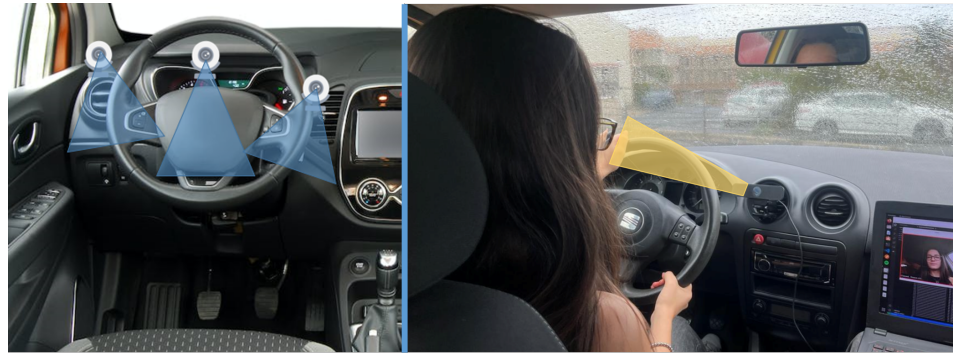
The aim of this study is two-fold. Firstly, a program was developed to perform facial recognition of a driver when they attempt to start the vehicle to verify whether that person is authorized to drive it or not. In order to achieve this, the Python [24] library for face recognition was used, which calculates the facial embedding of the driver's face and compares it with known and authorized faces. Secondly, a phase of research was carried out on the various existing approaches to detect driver drowsiness, identifying three main types of techniques for the detection of driver drowsiness based on the physiological characteristics of drivers, artificial vision techniques for driver tracking, and vehicle driving patterns. On the one hand, in this research, the method based on vehicle driving patterns was discarded due to the cost factor, as one of the main objectives was to create a system that could be used in any vehicle. On the other hand, this work focuses on driver identification using video and artificial vision techniques to track the driver and detect drowsiness, specifically, taking the face, eyes, and mouth as key points for driver identification and drowsiness detection.

The main technologies used for facial recognition are detailed below. In particular, the implementation uses the Python face recognition library [25] to process detected faces and calculate facial embeddings. In order to detect a driver's face, OpenCV [26], which is an artificial vision library that not only detects faces but also identifies their components, such as eyes, nose, and mouth, was used. As for the requirements of this part, it was necessary that at the time of purchasing the vehicle, the dealership personnel or a specialized company load the facial biometric data of authorized individuals along with their names. In this way, each time an attempt is made to start the vehicle, the camera inside will be activated, and the data stored in the database of authorized drivers will be compared to the set of facial features of the person present in the vehicle at that moment.

In the drowsiness detection phase, the main goal was to detect the face and eyes to identify characteristic signs of sleepiness, such as prolonged eye closure. Additionally, to perform this task, ML was used, specifically a model built using a CNN, where the proposed system was implemented using the Python language, the deep learning framework Keras [27], and the computer vision library OpenCV.

This work has several pre-requisites. Firstly, there must be a camera inside the vehicle to capture images of the driver through a Jetson Nano (see Figure 2). Secondly, several software tools are required. For instance, Python, the programming language used for development, must be installed to use the necessary packages. Furthermore, the open-source computer vision library OpenCV is required for face and eye detection, the framework Keras is required for creating the classification model, and the cross-platform Pygame [28] Python modules for playing the alarm sound that wakes up the driver are also needed.

Finally, the server makes use of Docker [29] to encapsulate an application and its dependencies in an isolated and portable environment, facilitating its deployment and distribution in different runtime environments. It provides greater flexibility and efficiency in server resource management and simplifies application deployment because this approach involves encapsulating both code and dependencies in a self-contained container. Moreover, compatibility issues between environments were reduced, ensuring that the application runs the same way on any server where it is deployed.



**Figure 2.** Possible camera placement options and selected final camera placement inside the car.

### 3.1. In-Car Camera

One of the key hardware requirements for this work is the placement of the camera inside the car. The specific position of the camera is a highly relevant factor for accurately measuring both the facial embedding and drowsiness levels through facial features, such as eye condition.

Figure 2 shows the three different camera angles, on the left, in the center, and on the right side of the steering wheel, that were tested during the experimentation for this work. Based on the experimental study, it was concluded that the left-side camera angle provides poor visibility of the driver's eyes, with significant interference from the driver's hand within its field of view. The center camera is positioned too closely to the steering wheel due to its central location, which necessitates placing the camera too high and could hinder road visibility. On the other hand, the right-side camera proved to be effective in detecting the driver's face and eyes. Being positioned above the air conditioning system neither obstructs the field of view of the road nor gets too close to the driver's face. Camera placements on the car's central rearview mirror and the top of the dashboard were also tested but were not considered suitable for safety reasons, as they obstructed the driver's vision. In conclusion, as shown in Figure 2, the optimal angle for detecting the driver's face and eyes is to place the camera on the right side of the steering wheel using the air conditioning vent hole.

### 3.2. Convolutional Neural Networks

Once the vehicle is started and the driving begins, the monitoring of relevant facial features is initiated using the camera. In order to achieve this, image processing techniques are employed to determine the level of drowsiness, typically through the application of machine learning techniques. Some of these techniques are convolutional neural networks (CNNs), support vector machines (SVMs) [30], and hidden Markov models (HMMs) [31]. In order to create a drowsiness detection model, one of those three types of techniques must be trained using labeled features and outcomes. As mentioned earlier, the models applied in this work use a CNN, which is an exclusive class of deep neural networks that perform exceptionally well when used for image classification.

A CNN includes essential components such as an input, an output, and a hidden layer with the potential for many layers. The input layer receives all inputs, whereas the last layer is the output layer, which provides the desired output. All intermediate layers are referred to as hidden layers. The hidden layers depend on the analyzed use case. These layers are convolved through a filter that multiplies the 2D matrices of the layer and the filter. The final layer has two nodes and is also fully connected.

The following layers constitute the unique architecture of the CNN model that was used:

- Two layers of 32 nodes, with a kernel size of 3;
- A total of 64 nodes in the convolutional layer, with a kernel size of 3;
- A total of 128 nodes for the fully connected layer.

The rectified linear unit (ReLU) [32] function is commonly used as the activation function for hidden layers in CNN models. However, in this work, the ReLU function was not applied to the output layer but was used in all the other layers. Instead, the Softmax function was applied just before the output layer.

### 3.3. Other Devices

Initially, this project was conducted using a MacBook Pro computer equipped with an Apple M1 Pro chip and 16 GB of random-access memory (RAM). Both driver identification and the drowsiness detection algorithms were successfully tested on this device in real time, demonstrating its suitability for the project's goals. However, considering the impracticality of using a laptop while driving, alternative devices that are smaller, more convenient, secure, and do not obstruct the driver's visibility were explored.

As a result, it was considered more appropriate to establish a client-server scheme in which the NVIDIA Jetson Nano Developer Kit 2 GB [33] serves as the client, and the server is dockerized on an external machine. Additionally, a second client was available at Nokia's company offices, acting as a control client. This is because the Jetson Nano only has 2 GB of graphics, which made the initial tests difficult as this was insufficient for analysis in real time, but it can work in situations such as a loss of coverage with less image capture frequency.

The decision to adopt this architecture stemmed from the constrained graphical capacity of the Jetson Nano, which posed challenges during initial tests. With only 2 GB of graphics memory, real-time analysis proved inadequate. However, the device exhibited satisfactory performance under circumstances involving sporadic image captures.

In order to quantify the efficacy of the driver face recognition program, the average execution time was measured. These figures represent the average total execution time across tests conducted under optimal conditions, wherein the user's face was unobstructed by glasses, masks, or hair. Notably, the computational overhead increases with a larger database of individuals, potentially impeding program responsiveness. Nonetheless, this is unlikely to pose a significant issue in practical scenarios, given that vehicles typically accommodate one to three occupants.

Furthermore, the execution time is contingent on the processing capabilities of the device, with superior performance observed on higher-spec machines. Consequently, the server was selected as the preferred platform for frame analysis, surpassing the computational capabilities of the Jetson Nano.

A comparison of average execution times for driver face recognition is provided in Table 1. Notably, face recognition processing on the server yielded an average time of 3.84 s when gazing at the camera and 8.79 s when focused on the road. In contrast, analysis conducted on the Jetson Nano exhibited significantly longer processing times, averaging 107.51 s and 106.96 s, respectively.

Regarding drowsiness detection, the server achieved a processing rate of 0.15–0.16 frames per second (FPS), demonstrating satisfactory performance. Conversely, the latency experienced on the Jetson Nano was prohibitively high, rendering drowsiness detection impractical.

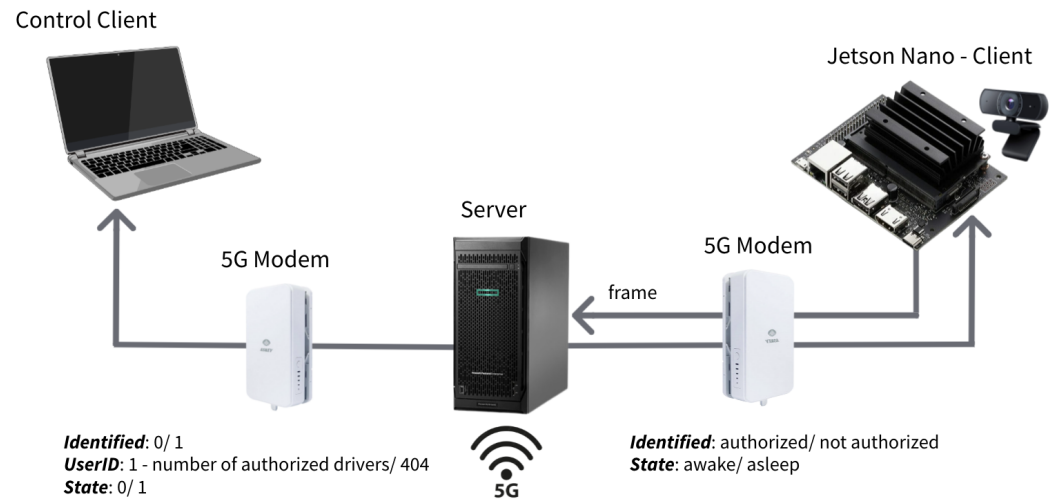
These findings underscore the inadequacy of the Jetson Nano for real-time analysis without substantial system modifications.

**Table 1.** Average facial recognition time

Driver's View	Server Average Time	Jetson Nano Average Time
At the camera	3.84 s	107.51 s
At the road	3.79 s	106.96 s

#### 4. System Architecture

The system architecture features a client-server scheme with a single server and two clients, as depicted in Figure 3. Initially, the program was tested on a laptop without the client-server scheme, running solely within a single file.



**Figure 3.** Client-server scheme.

There is the Jetson Nano client located inside the car with a camera focusing on the driver. The main program runs on this client, opening the camera and sending captured images of the driver in the car to the server or processing the image in case of a loss of coverage. The server houses the part of the program that analyzes the image data received from the Jetson Nano client. As for driver identification, features that allow for the recognition of a person are extracted and compared to the characteristics in the database containing all authorized drivers to start the vehicle. If the ID of the person received from the client is in the Server database, he/she is authorized; otherwise, it would be considered an attempted theft, and the user will not be authorized to start the car. Note that the Server database does not contain all the user's data, only the IDs. In this way, the privacy of drivers cannot be compromised in the case of drowsiness detection. For drowsiness detection, typical signs of sleepiness are determined by identifying the person's face and eyes, relying on the height of the eyes to determine if they are closed or open. When both eyes are closed for more than a certain period (greater than a simple blink to avoid false positives), the driver's state is classified as asleep. Otherwise, the driver's condition remains awake. If the Server (or the Jetson Nano client in case of loss of coverage) detects a drowsiness state, i.e., the driver is asleep, an alarm is triggered to awaken them.

In addition to this, the control client is responsible for receiving performance key indicators from the Server, including the user number inside the vehicle and the driver's state. Regarding identification, this will have a value of 0 if the user is authorized to start the car and 1 if they are not authorized. In the case that the person is not authorized, the user number is 404. Finally, the driver's state has a value of 0 if they are awake and 1 if they are asleep.

The entire project architecture is implemented using 5G networks. For this purpose, two 5G modems were used, each connected to one client, one to the Jetson Nano and the other to the control client. The use of 5G networks has the advantage of offering an ultra-fast connection speed, which enables instant data transfer, facilitating the download and real-time streaming of high-quality multimedia content, such as 4K or even 8K videos. Additionally, the low latency of 5G ensures an instant response in online applications and services, which are crucial aspects of the project that was carried out.

## 5. Implementation

This section details the explanation of the code used in each of the architecture components described in the previous section to better understand the specific functionality of each element.

### 5.1. Server

The server includes different functions for key performance indicators (KPIs) [34], driver ID, drowsiness monitoring, sending statistic data to the control client, and the main function.

#### 5.1.1. KPI Function

The first class of the server.py file contains the functions in charge of sending reports or metrics to a specific destination using the user datagram protocol (UDP). The first of these is `__init__()`, which receives a series of parameters, including `addr`, which contains the destination IP; in this case, it is 127.0.0.1, referring to local use. The next one is the port, which is configured with 8094, which, as previously mentioned, is the port in charge of providing the UDP service. Within this method, the variables that it receives as parameters are instantiated in addition to creating a socket object and storing it in `self.socket`. Finally, all additional arguments that the function may receive are stored in the `self.tags` dictionary.

Second is the `tag()` method, which is responsible for adding or modifying tags to the report. It receives keyword arguments, `**kwargs`, which are treated as key-value pairs and updated in the `self.tags` dictionary. Then, the `self`-object is returned, which allows for chaining multiple calls to the `tag()` method.

The third function is `report()`, which is used to send the report or metric to the specified destination. Inside it, a text string is constructed containing the name of the `self.measurement`, the `self.tags`, and the `kwargs` values. If a timestamp, `ts`, is provided, it is appended to the end of the text string. If the value `self.v` is `True`, the text string is printed to the console using `print(txt)`. Finally, the UDP socket `self.socket` is used to send the byte-encoded report to the specified destination, thanks to `self.addr` and `self.port`.

#### 5.1.2. Facial Recognition Function

This code section is responsible for facial recognition in drivers. In general terms, it receives images from the client inside the vehicle, analyzes them, searches for matches among authorized individuals' faces, and sends a signal to the vehicle to start or not.

The code starts with the definition of the `state_recognition` function, which takes four parameters:

- `img`: This parameter stores the current frame of the real-time camera;
- `name_prove`: This variable stores the name of the person who wants to start the vehicle at that moment. Later, it will be compared with `classNames` to see if it has permissions;
- `classNames`: This is in charge of storing the names of the people authorized to start the vehicle;
- `authorizeEmbeddings`: This parameter stores the facial embeddings of the people authorized to drive the vehicle.

Next, the current frame captured by the camera is stored in the `im` variable, transforming it to a format suitable for processing. A transformation is performed to adjust the range of pixel values from  $[-1, 1]$  to  $[0, 255]$  and is converted to `uint8`. Two empty lists are initialized: `faceDistances`, which will store the Euclidean distance when comparing the face in the current frame with all authorized faces, i.e., how similar the faces are, and `permission` to store the authorization status. The `facesCurrFrame` variable stores the matrix of bounding boxes for the faces found in that frame, i.e., their location, whereas `encodesCurrFrame` stores the vector of 128 key points that make each face unique.

In this step, the current frame is compared to all images of the authorized individuals to find any matches, which will be stored in `'matches'` using the `compare_faces` function. If the vector of 128 points differs by 0.6 or less from the default value, it means both faces



are considered a match. Therefore, all face vectors that meet this condition are stored in `faceDistances`. If there is a match, the authorized person's name is stored in the `'name'` variable, but the output is formatted differently. By default, the images have the format `"First_Last.jpg"`; therefore, this function changes this format to `"First Last"`. Finally, the words `"Authorized"` or `"Refused"` are stored in the `'permission'` variable. This will be used to send the starting signal to the vehicle (or not), in addition to other uses for KPIs.

In relation to facial recognition, Figure 4 depicts the outcome of a run, illustrating a vector of facial embeddings stored in the vehicle's database. The individual facing the camera corresponds to the first value of the facial embedding vector, displaying a value closest to 0, indicating the degree of similarity to the individual. A value approaching 1 suggests no resemblance to the person in front of the camera. It is evident across multiple iterations that the individual is consistently recognized correctly, thereby confirming their identity and authorization to initiate the vehicle. This numerical representation signifies the accuracy of resemblance between the individual facing the camera and those authorized to drive, serving as a metric to assess the quality of facial recognition.

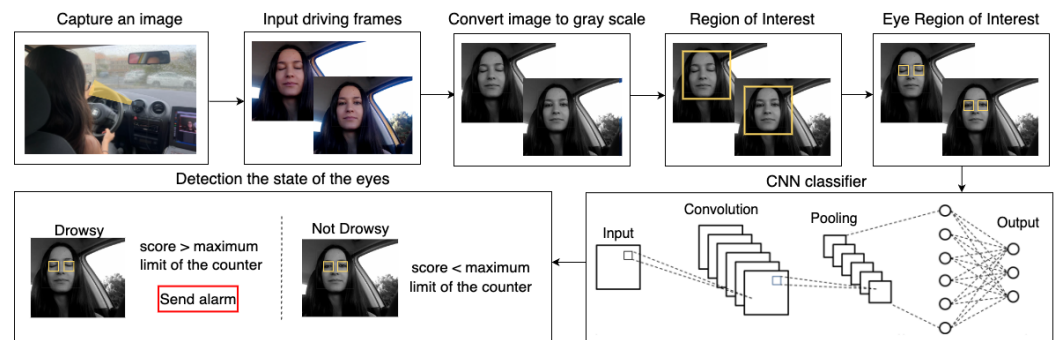
```
[0.35809964 0.77555804 0.81214557 0.73190681 0.90978466]
OSCAR CIGALA ALVAREZ
[0.35811553 0.78058865 0.83136761 0.74122779 0.91424272]
OSCAR CIGALA ALVAREZ
[0.35585836 0.78352279 0.83207123 0.74400698 0.91398353]
OSCAR CIGALA ALVAREZ
[0.34990045 0.79486364 0.82871493 0.73438374 0.92308679]
OSCAR CIGALA ALVAREZ
[0.35218204 0.78975701 0.81460998 0.72256095 0.91867517]
OSCAR CIGALA ALVAREZ
[0.3591092 0.79197899 0.82535937 0.73851042 0.90506731]
OSCAR CIGALA ALVAREZ
[0.34542079 0.80002845 0.81202501 0.72018181 0.92249729]
OSCAR CIGALA ALVAREZ
[0.35181394 0.79479195 0.81331909 0.72683178 0.9277594 ]
OSCAR CIGALA ALVAREZ
[0.34916812 0.77911708 0.82658966 0.71950227 0.92360426]
OSCAR CIGALA ALVAREZ
[0.34257954 0.78157542 0.82390061 0.71531453 0.92129927]
OSCAR CIGALA ALVAREZ
True
```

**Figure 4.** Result of the facial recognition code.

### 5.1.3. Drowsiness Detection Function

In the drowsiness monitoring section, an ML model is employed in conjunction with the OpenCV library to discern the driver's state of drowsiness by scrutinizing images acquired from a camera installed within the vehicle. The model proficiently categorizes the condition of the driver's eyes, classifying them as either `"open"` or `"closed"`. The dataset utilized for training this model has been sourced from DataFlair. The procedural methodology encompasses the creation of a dataset through the capture of images of the driver's eyes via a camera, followed by their classification into `"open"` or `"closed"` categories.

During the data cleaning phase, meticulous criteria are applied, involving the elimination of superfluous images, particularly those characterized by suboptimal quality or lacking relevance for model training. This curation process encompasses approximately 7,000 photographs, capturing human eyes in diverse environmental settings. Subsequently, the ML model undergoes a comprehensive training process. The conclusive weights and architecture file of the model, denominated `'cnnCat2.h5'`, are enclosed within the `'models'` folder. The resultant model proves instrumental in ascertaining whether the driver's eyes are open or closed, thereby facilitating the identification of indications of drowsiness during the act of driving. The sequential steps employed for the detection of driver drowsiness are delineated as follows and are shown in Figure 5.



**Figure 5.** Architecture of the proposed methodology for drowsiness detection.

1. **Capture an image through the camera:**  
A webcam is used to capture an image for input. An infinite loop is constructed to record each frame from the camera. The camera is accessed, and the capture object cap is set using the OpenCV method `cv2.VideoCapture(0)`. Each frame is read with `cap.read()`, and the image is stored in a frame variable.
2. **Creation of a region of interest (ROI) by detecting faces:**  
In order to detect faces in an image, the image is converted to gray scale since the OpenCV object detection algorithm accepts gray scale images as input. In order to find faces, a cascaded classifier is used. Next, `faces = face.detectMultiScale(grey)` is used for detection. It returns an array of hits containing the *x*, *y*, height, and width co-ordinates of the bounding box of the object. Now, it is possible to iterate over the faces and draw bounding boxes for each face.
3. **Determining the eye region of interest (ROI) and providing it to the classifier:**  
Eyes are discovered using the same technique as for the faces. The left eye (*leye*) and right eye (*reye*) cascade classifiers are initialized, and the `detectMultiScale(gray)` function is used to detect the eyes. Then, only the eye data are extracted from the entire image. This is accomplished by removing the eye-bounding box. The variable *l\_eye* contains the image data for the left eye and *r\_eye* for the right eye. This is fed into a CNN classifier, which predicts whether the eyes are open or closed.
4. **The classifier detects whether the eyes are open or closed.**  
A CNN classifier is used to predict the state of the eyes. First, the color image is converted to a gray scale. Then the image is resized to  $24 \times 24$  pixels. The code `lpred = model.predict_classes(l_eye)` can be used to predict each eye using the trained model. The `predict_classes` function returns the class 0 or 1, as predicted by the model for the given input image. If the value of `lpred[0]` is equal to 1, it indicates that the eyes are open. On the other hand, if the value of `lpred[0]` is equal to 0, it means that the eyes are closed.
5. **It calculates a score to determine if the subject is asleep:**  
A score is essentially a value used to determine how long a person keeps their eyes closed. Therefore, if both eyes are closed, the score will always increase, and if both eyes are open, the score will decrease.  
The sleep detection function must receive the following variables:
  - (a) **img:** The image received from the customer of the Jetson Nano, i.e., an image of the driver.
  - (b) **upper\_bound:** This is the score that specifies the maximum limit of the counter for determining whether the driver has fallen asleep.
  - (c) **score:** This is the variable that starts increasing its value when the person closes his eyes until he opens them again. If this value reaches the same value as the upper-bound variable, it is determined that the person has fallen asleep.

The Haar cascade classifier is an ML-based approach that is typically used with many positive and negative images for training. In this work, it was used for the face and eyes. The model is loaded to predict each eye. Then, the “state” variable is created to check the

state of the eyes (open/closed). The right eye is detected. In order to do this, only the eye data are extracted from the full image, selecting the eye bounding box and then extracting the eye image from the frame with this code. Afterward, the color of the image is converted into grayscale. Subsequently, the image is resized to  $24 \times 24$  pixels since the model was trained using that image size. Then, the data are normalized for better convergence, with the data values between 0 and 1. If the value of "rpred[0]" = 1, it indicates that the eyes are open; if the value is 0, then it is concluded that the eyes are closed. The monitoring of the left eye is done in the same way; if both eyes are closed, the score counter increases. When both eyes are open, the score decreases. If the score is higher than the value of the upper\_bound variable, it means that the person's eyes have been closed for a long time. Once this function is finished, the driver's state (awake/asleep) is sent to the Jetson Nano client. If the state is asleep, the alarm is triggered to wake them up and prevent a potential accident.

#### 5.1.4. Client Control Function

This server function is intended to send the control client the resulting image of the analysis, i.e., the square marking the boundaries of the person's face, the name of the authorized driver of the vehicle, and their drowsiness status.

First, the driver's face is detected. Later, if the person attempting to start the car is not an authorized individual to do so, it means that they are not in the vehicle's database and, therefore, are not identified. Consequently, they are given the name UNKNOWN, with the user identifier 404. If the person is authorized, their data are recorded. The variables per and st, which stand for permission and state, are created with their default values, where the driver's state is awake, and they are not authorized.

Next, the bounding box around the person's face is drawn, and their data are added. In this step, the values of the variables per and st are changed based on the driver's state. If the driver is awake, both variables have a value of 0. If they are asleep, the value of the st variable changes to 1. In the case of the permission variable (per), if the driver is authorized, its value is 0, and if they are not authorized, it is 1.

Finally, the image is resized, and the package information, the identification variable value, the user identifier, and the state variable of the driver's condition are sent to the control client.

#### 5.1.5. Main Function

The run\_segmentation\_test() function is the main method that executes the program. It receives several parameters, such as inaddr, picture\_outaddr, and json\_outaddr, which specify the ports to send or receive data.

Next, some contexts are established using the PictureInput, PictureOutput, and JsonOutput objects. These contexts allow for the reading of input images, the output of processed images, and the output of JSON metadata, respectively.

Several variables are initialized, including metadata, classNames, authorizeEmbeddings, name\_prove, score, THRESHOLD, cont, and id. Additionally, the threshold value (upper\_bound) is read from the './Data/score.txt' file. In addition, the './Data/score.txt' folder is accessed, which contains values that can be modified without having to create another Docker specifically for it. In this section, the output format of the output\_picture.format image is set to jpeg, and a packet object is created to send images in U8Picture format. The names of all persons who have permits to drive the vehicle are printed on the screen.

Afterward, the face recognition part begins. The cont\_auth variable is initialized and a "while" loop is executed for the recognition stage. The loop is executed as long as cont is greater or equal to 1; by default, it starts with 10. In this way, the driver of the vehicle is authenticated 10 times to be sure that this person has permission to start the vehicle. Then, the images received from the vehicle camera are stored in img. The default status is "Awake" because the car has not been started yet and it does not matter if the driver is asleep. Permission stores the status 0 if the person is authorized and 1 if not.

The next section of code shows the drowsiness functionality, in which an image is received, the drowsiness detection function is called, the information is sent to the control client, and the driver's status is displayed, which can be either asleep or awake. Finally, the connection ports between the server and the two clients are established.

### 5.2. Jetson Nano Client

The following code sections belong to the code that runs the Jetson Nano Developer Kit 2 GB, more specifically, the file called `detection_client.py`, which simulates the software installed in the car. It is also responsible for running the camera and sending the images to the server, as well as receiving certain parameters, including whether the driver has permission to drive that vehicle, in which case it starts the engine, or if he is falling asleep, sounding the alarm to wake him up.

In the first lines of code, the libraries used are imported. The `pygame.mixer` is in charge of the alarm, whereas `cv2` is in charge of the camera. The `run` function contains the main code. In the beginning, it sets the video format to be sent to the server.

Subsequently, we run the camera and set the image size. A counter is set, called `conta`, to ensure that the facial recognition code is executed 10 times. This is done to make sure that if the person driving is authorized, he/she can start the vehicle without problems. The packets are then sent to the server and the response is received from the server to know if the engine has started or not.

Once the startup has been successful, the drowsiness part begins. Again, packets with the images captured by the camera are sent, but this time, whether the person is asleep or not is received from the server. If so, the vehicle alarm sounds to alert the driver.

### 5.3. Client Control

The control client is responsible for displaying (in a window) who is driving and whether they are asleep or awake. It is used only as a visual form of what the server is sending. In order to accomplish this, a box is created over the driver's face, and his name is added, as well as whether he is falling asleep or not. Finally, the connection port between the server and the client is established.

## 6. Results

The final results obtained were recorded in a demo at Nokia's facilities, showing the operation of the client-server architecture. As it is a tool that has to function in real time, the response times are minimal. This demo was carried out in a simulated environment so as not to put people, infrastructure, or other collateral damage at risk. For this, a seat was used as the driver's seat, and various car simulation devices, such as the steering wheel, gearshift, pedals, and the camera were located in the optimal position to record the driver's face. This test environment is shown in Figure 6, where the two 5G modems previously discussed can be seen.

The demo showed part of the results discussed above on the Jetson Nano client. Figure 7 shows how an unauthorized person fails to find a match within the vehicle database, preventing them from starting the vehicle. Furthermore, the results of the user identification section are displayed, including the user's name and their eye state, as shown in Figure 8.

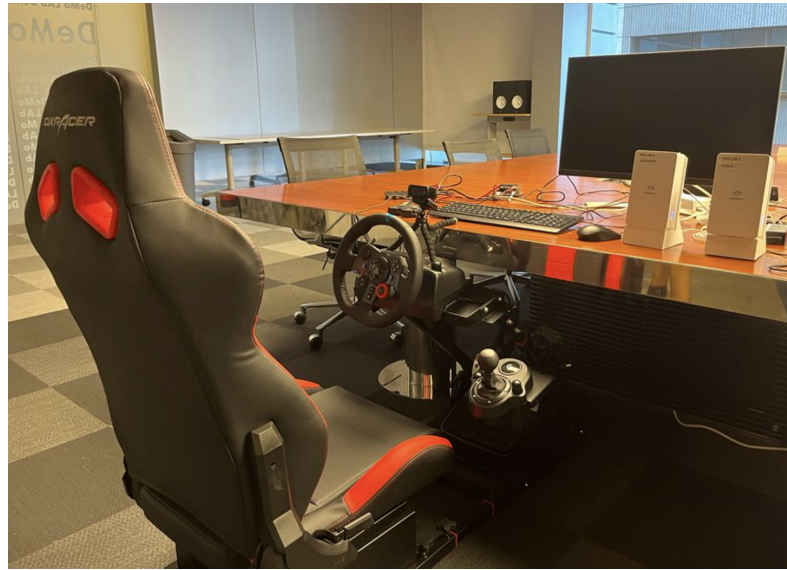


Figure 6. Test environment.

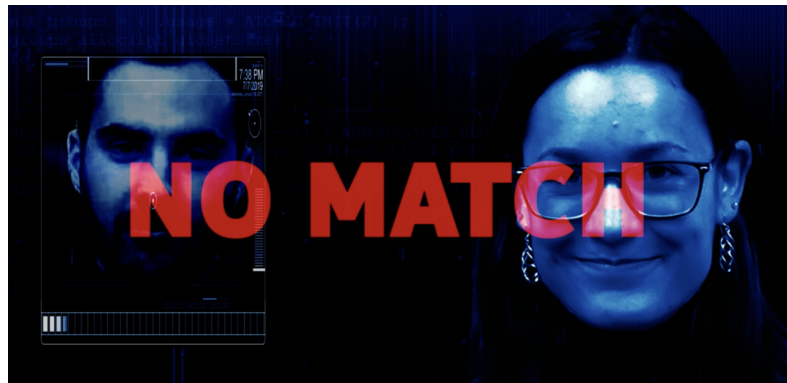


Figure 7. Result of unauthorized person trying to start the vehicle.



Figure 8. Person state result. Eyes open/closed.

For the control client, the KPI for authorized and unauthorized users, as well as driver status monitoring (asleep/awake), are presented. Figure 9 shows a user with identifier 404, which means that he is an unauthorized user who cannot start the car, leading to an instant program termination. The figure also shows a user with identifier 3, indicating an authorized user. The driver's identification variable is set to 0, allowing the user to start

the car. After the identification, real-time drowsiness monitoring begins, and the sleep state variable toggles between 0 (awake) and 1 (detecting drowsiness).



**Figure 9.** Authorized/Unauthorized person result.

Given the results obtained, we have achieved the proposed objectives and generated a program with client-server architecture that can be placed in any vehicle to be accessed only by authorized persons and in which drowsiness monitoring is performed in real time to avoid not only material damage but also accidents and physical damage.

In order to analyze the amount of data sent by the client to the server, the `iftop` command was used to test different configurations in the client code to see how it affects network consumption. Mainly, two parameters were modified: the number of frames per second that the client sends to the server, varying between 30, 15, and 5 FPS, and the resolution of the image captured by the camera, which is  $1280 \times 480$  pixels, maximum, and  $640 \times 240$ , minimum. As can be seen in Table 2, the value of megabits per second (Mbs) sent by the client varies between 22.9 Mbs and 1.08 Mbs. In order to avoid network saturation, it was decided to leave a default image resolution of  $1280 \times 480$  and 5 FPS.

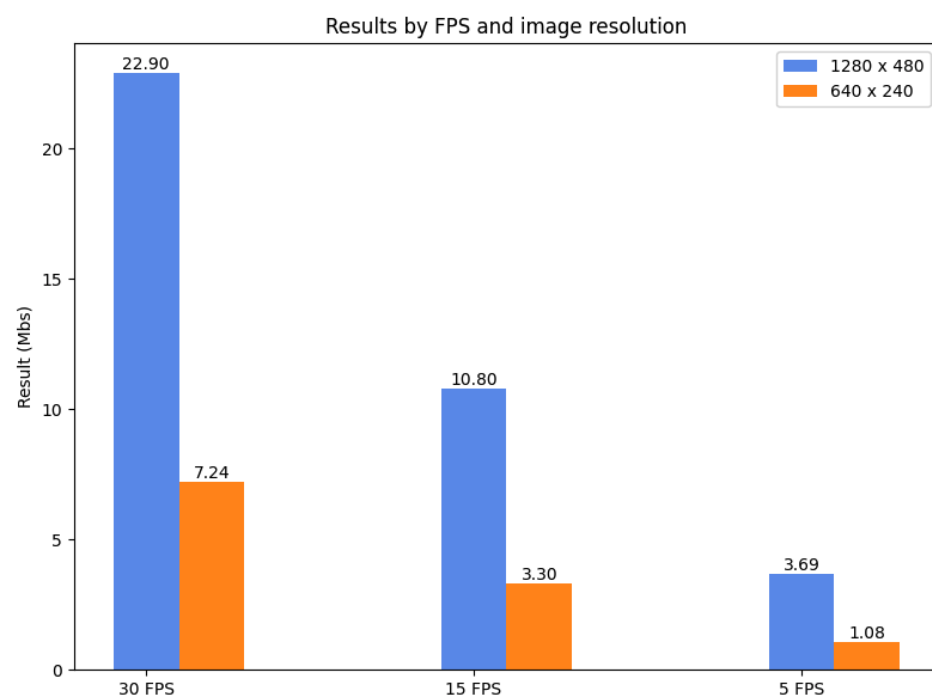
**Table 2.** Results obtained with iftop.

Frames per Second (FPS)	Image Resolution	Result
30	1280 × 480	22.9 Mbs
15	1280 × 480	10.8 Mbs
5	1280 × 480	3.69 Mbs
30	640 × 240	7.24 Mbs
15	640 × 240	3.3 Mbs
5	640 × 240	1.08 Mbs

In Figure 10, the bar chart effectively visualizes the relationship between FPS, image resolution, and network consumption in megabits per second (Mbs), as derived from the client-server data transmission tests. Each pair of bar clusters represents a specific image resolution 1280 × 480 pixels and 640 × 240 pixels. Within each cluster, individual bars correspond to different FPS settings: 30, 15, and 5 FPS. The height of each bar indicates the amount of data transmitted at that particular combination of resolution and FPS. This graphical layout allows for an immediate visual comparison across different configurations, highlighting the proportional increase in network usage with higher FPS and the more considerable data demands of the higher resolution setting. It serves as a compelling visual tool to complement the tabulated data, making the patterns and relationships in the data readily apparent and easily comprehensible.

In terms of system limitations, various aspects have been considered. In order to assess the effectiveness of the identification phase, the data collected were analyzed to gauge test success. These data were categorized into three outcomes:

1. Hit: Correct recognition of the driving individual by the program.
2. Mis-identifying: Failure of the program to authenticate any stored faces or detection of an incorrect driver.
3. Failure: The program is unable to detect any face, resulting in automatic closure.

**Figure 10.** Graphical iftop results.

A notable aspect of the tests pertains to the direction of the driver's gaze. Participants were instructed to alternate their gaze between the road and the camera to simulate real driving conditions. An analysis of results reveals variations depending on the driver's gaze direction. When looking at the road, the visibility of the left side of the driver's face may be compromised, significantly impacting facial embedding calculation. However, under ideal conditions with no facial obstructions, a 100% accuracy rate was achieved both when facing the camera directly and when focusing on the road.

The second test involved drivers wearing eyeglasses, resulting in a 97.5% recognition rate when facing the camera, with a 2.5% error rate. However, recognition dropped to 88% when drivers looked at the road, as 12% were misidentifying due to factors previously discussed, along with potential reflections in glasses hindering eye detection.

The subsequent tests involved drivers wearing sunglasses, yielding favorable results of 85.4% correct identifications and 14.6% wrongly recognized when facing the camera. However, recognition dropped to 49.2% when drivers looked at the road, with 36.2% instances of misidentifying and 14.6% failures, primarily due to obstructed facial features.

The tests involving masks showed expected outcomes, with recognition rates significantly lower due to facial coverage. When facing the camera, the recognition rates were 23.6% for hits, 20.7% for wrongly recognized, and 55.7% for misses. When looking at the road, recognition was unsuccessful in 100% of cases due to the combined effects of mask coverage and obscured facial features. Finally, tests involving drivers with long, loose hair showed 100% accuracy when their hair was tied back, demonstrating negligible impact on facial recognition.

For the drowsiness phase, algorithm robustness was initially characterized using various videos featuring 15 individuals in different conditions. The tests were conducted with subjects seated in a car, with the camera positioned consistently. Factors such as wearing glasses, masks, hair obstruction, and lighting conditions were considered.

The initial challenges included algorithm malfunction under low illumination, difficulty distinguishing closed eyes from half-closed eyes due to light, and misreading eyes during face turns. Notably, the tests did not involve sunglasses, as drowsiness accidents are less likely in bright conditions. Despite these challenges, the detection system yielded 100% accuracy.

In order to address these limitations, future iterations could incorporate high-quality infrared or thermal cameras to enhance eye detection under challenging conditions such as low light or when wearing sunglasses.

## 7. Conclusions

This work reflects the culmination of a research project that has produced significant technological advancements with practical applications in vehicle security and driver safety, leveraging cutting-edge technologies, including 5G/6G, edge computing, and real-time data processing. The project's demonstration at Nokia's facilities showcased the effectiveness of the client-server architecture. Importantly, the demonstration was conducted in a simulated environment to ensure the safety of individuals, infrastructure, and other road users.

The setup involved a simulated car with all the essential driving controls, including a camera optimally positioned to record the driver's face, and two 5G modems, as discussed in the architecture section, enabling high-speed data transmission and low latency communication.

One notable achievement was the successful implementation of facial recognition to verify the driver's authorization. The tests demonstrated that unauthorized individuals could not start the vehicle, effectively addressing vehicle security concerns through the use of advanced 5G connectivity and edge computing capabilities. Additionally, attaining a significantly low execution time for the facial recognition component enhances effectiveness and efficiency without compromising driver comfort or time expenditure.



Furthermore, the real-time monitoring of driver drowsiness, as discussed in previous sections, was also validated during the demonstration. This capability holds immense promise for preventing not only material damage but also accidents and physical harm caused by drowsy driving, made possible by the real-time data processing capabilities of the system.

Finally, the system underwent a comprehensive assessment encompassing speed, latency, and performance. The outcomes substantiate the effective visibility of the individual's face for person recognition and authorization to operate the vehicle. Moreover, the algorithm adeptly assesses the driver's drowsiness in real time, achieved through the adjustment of frames per second parameters and image resolution during data transmission from the client to the server. Leveraging the capabilities of 5G networks ensures a more streamlined and efficient real-time data transmission, eliminating latency issues while sending frames promptly.

In order to facilitate real-time execution on the Jetson Nano platform and achieve acceptable inference times, future work should focus on either enhancing the data training process or optimizing the system architecture. Additionally, a comprehensive investigation into server infrastructure requirements, including server quantity and processing capacity, is necessary for large-scale deployment within vehicles. This deployment scenario necessitates a robust server infrastructure capable of handling the periodic transmission and processing of images for driver monitoring tasks, such as fatigue detection.

**Author Contributions:** Conceptualization, E.G.-S., P.C.-G. and C.C.-G.; Methodology, S.D.-S., Ó.C.-Á., E.G.-S., P.C.-G. and C.C.-G.; Software, S.D.-S., Ó.C.-Á. and E.G.-S.; Investigation, S.D.-S., Ó.C.-Á. and E.G.-S.; Resources, S.D.-S. and E.G.-S.; Writing—original draft, S.D.-S. and Ó.C.-Á.; Writing—review & editing, P.C.-G. and C.C.-G.; Supervision, P.C.-G. and C.C.-G.; Project administration, P.C.-G. and C.C.-G.; Funding acquisition, P.C.-G. and C.C.-G.. All authors have read and agreed to the published version of the manuscript.

**Funding:** This Research was supported by the CDTI (Centre for the Development of Industrial Technology), the Ministry of Economy Industry and Competitiveness, Celtic-Plus EUREKA, and the European Regional Development Fund, under Project IMMINENCE C2020/2-2 and the European Union (Next Generation) under the strategic project C064/23 SCITALA.

**Informed Consent Statement:** Informed consent was obtained from all subjects involved in the study.

**Data Availability Statement:** The raw data supporting the conclusions of this article will be made available by the authors on request.

**Conflicts of Interest:** Author Ester Gonzalez-Sosa was employed by the company eXtended Reality Lab. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

IoT	Internet of Things
ITS	Intelligent Transport Systems
AI	Artificial Intelligence
DGT	Dirección General de Tráfico
GSM	Global System for Mobile Communications
GPS	Global Positioning System
SMS	Short Message Service
ML	Machine Learning
CNN	Convolutional Neural Networks
SVM	Support Vector Machines
HMM	Hidden Markov Models
ReLU	Rectified Linear Units
GB	GigaByte

RAM	Random-Access Memory
KPI	Key Performance Indicators
UDP	User Datagram Protocol
ROI	Region of Interest
FPS	Frames Per Second
Mbs	Megabits Per Second

## References

1. EU General Safety Regulation. Available online: <https://www.continental-automotive.com/en/industry/trucks-and-buses/eu-general-safety-regulations.html> (accessed on 21 February 2024).
2. Perkins, E.; Sitaula, C.; Burke, M.; Marzbanrad, F. Challenges of driver drowsiness prediction: The remaining steps to implementation. *IEEE Trans. Intell. Veh.* **2022**, *8*, 1319–1338. [[CrossRef](#)]
3. Avance de las Principales Cifras de la Siniestralidad Vial. 2020. Available online: [https://www.dgt.es/export/sites/web-DGT/galleries/downloads/dgt-en-cifras/24h/Las-principales-cifras-2020\\_v6.pdf](https://www.dgt.es/export/sites/web-DGT/galleries/downloads/dgt-en-cifras/24h/Las-principales-cifras-2020_v6.pdf) (accessed on 12 January 2024).
4. Kosalendra, E.; Leema, G.; Muni, V.P.K.; Kartheek, I.; Hemanth, K.C. Intelligent Car Anti-Theft System Through Face Recognition Using Raspberry Pi and Global Positioning System. *Int. J. Anal. Exp. Modal Anal.* **2020**, *12*, 1017–1021.
5. Arduino-Home. Available online: <https://www.arduino.cc/> (accessed on 19 February 2024).
6. Sanda, P.; Barui, S.; Das, D. SMS Enabled Smart Vehicle Tracking Using GPS and GSM Technologies: A Cost-Effective Approach. In *Smart Systems and IoT: Innovations in Computing: Proceeding of SSIC 2019*; Springer: Singapore, 2020; pp. 51–61. [[CrossRef](#)]
7. Li, Z.; Chen, L.; Peng, J.; Wu, Y. Automatic Detection of Driver Fatigue Using Driving Operation Information for Transportation Safety. *Sensors* **2017**, *17*, 1212. [[CrossRef](#)] [[PubMed](#)]
8. Gwak, J.; Hirao, A.; Shino, M. An Investigation of Early Detection of Driver Drowsiness Using Ensemble Machine Learning Based on Hybrid Sensing. *Appl. Sci.* **2020**, *10*, 2890. [[CrossRef](#)]
9. Bangui, H.; Cioroica, E.; Ge, M.; Buhnova, B. Deep-Learning based Trust Management with Self-Adaptation in the Internet of Behavior. In Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, Tallinn, Estonia, 27–31 March 2023; pp. 874–881.
10. Wang, J.; Chai, W.; Venkatachalapathy, A.; Tan, K.L.; Haghghat, A.; Velipasalar, S.; Adu-Gyamfi, Y.; Sharma, A. A survey on driver behavior analysis from in-vehicle cameras. *IEEE Trans. Intell. Transp. Syst.* **2021**, *23*, 10186–10209. [[CrossRef](#)]
11. Trenta, F.; Conoci, S.; Rundo, F.; Battiato, S. Advanced motion-tracking system with multi-layers deep learning framework for innovative car-driver drowsiness monitoring. In Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition, Lille, France, 14–18 May 2019; pp. 1–5.
12. Ngxande, M.; Tapamo, J.; Burke, M. Driver drowsiness detection using behavioral measures and machine learning techniques: A review of state-of-art techniques. In Proceedings of the Pattern Recognition Association of South Africa and Robotics and mechatronics, Bloemfontein, South Africa, 29 November–1 December 2017; pp. 156–161.
13. Chirra, V.; Uyyala, S.; Kolli, V. Deep CNN: A Machine Learning Approach for Driver Drowsiness Detection Based on Eye State. *Rev. D'Intelligence Artif.* **2019**, *33*, 461–466. [[CrossRef](#)]
14. Tanveer, M.; Khan, M.; Qureshi, M.; Naseer, N.; Hong, K. Enhanced drowsiness detection using deep learning: An fNIRS study. *IEEE Access* **2019**, *7*, 137920–137929. [[CrossRef](#)]
15. Jabbar, R.; Shinoy, M.; Kharbeche, M.; Al-Khalifa, K.; Krichen, M.; Barkaoui, K. Driver drowsiness detection model using convolutional neural networks techniques for android application. In Proceedings of the IEEE International Conference on Informatics, IoT, and Enabling Technologies, Doha, Qatar, 2–5 February 2020; pp. 237–242.
16. Hashemi, M.; Mirrashid, A.; Beheshti Shirazi, A. Driver safety development: Real-time driver drowsiness detection system based on convolutional neural network. *SN Comput. Sci.* **2020**, *1*, 1–10. [[CrossRef](#)]
17. What are Convolutional Neural Networks? | IBM. Available online: <https://www.ibm.com/topics/convolutional-neural-networks> (accessed on 19 February 2024).
18. Dua, M.; Singla, R.; Raj, S.; Jangra, A. Deep CNN models-based ensemble approach to driver drowsiness detection. *Neural Comput. Appl.* **2021**, *33*, 3155–3168. [[CrossRef](#)]
19. William, P.; Shamim, M.; Yeruva, A.R.; Gangodkar, D.; Vashisht, S.; Choudhury, A. Deep Learning based Drowsiness Detection and Monitoring using Behavioural Approach. In Proceedings of the 2022 2nd International Conference on Technological Advancements in Computational Sciences (ICTACS), Chicago, IL, USA, 10–12 October 2022; pp. 592–599. [[CrossRef](#)]
20. Chand, H.V.; Karthikeyan, J. Cnn based driver drowsiness detection system using emotion analysis. *Intell. Autom. Soft Comput.* **2022**, *31*, 717–728. [[CrossRef](#)]
21. Kumar, V.; Sharma, S. Driver drowsiness detection using modified deep learning architecture. *Evol. Intel.* **2022**, *16*, 1907–1916. [[CrossRef](#)]
22. Phan, A.C.; Nguyen, N.H.Q.; Trieu, T.N.; Phan, T.C. An Efficient Approach for Detecting Driver Drowsiness Based on Deep Learning. *Appl. Sci.* **2021**, *11*, 8441. [[CrossRef](#)]
23. Chen, S.; Wang, Z.; Chen, W. Driver Drowsiness Estimation Based on Factorized Bilinear Feature Fusion and a Long-Short-Term Recurrent Convolutional Network. *Information* **2021**, *12*, 3. [[CrossRef](#)]
24. Welcome to Python.org. Available online: <https://www.python.org/> (accessed on 19 February 2024).

25. Face-Recognition. PyPI. Available online: <https://pypi.org/project/face-recognition/> (accessed on 19 February 2024).
26. Home. OpenCV. Available online: <https://opencv.org/> (accessed on 19 February 2024).
27. Keras: Deep Learning for Humans. Available online: <https://keras.io/> (accessed on 19 February 2024).
28. Pygame. PyPI. Available online: <https://pypi.org/project/pygame/> (accessed on 19 February 2024).
29. Docker: Accelerated Container Application Development. Available online: <https://www.docker.com/> (accessed on 19 February 2024).
30. 1.4. Support Vector Machines. Scikit-Learn. Available online: <https://scikit-learn.org/stable/modules/svm.html> (accessed on 19 February 2024).
31. Hidden Markov Models. Stanford University. Available online: <https://web.stanford.edu/~jurafrsky/slp3/A.pdf> (accessed on 19 February 2024).
32. Polo Club of Data Science @ Georgia Tech: Human-Centered AI, Deep Learning Interpretation & Visualization, C.L.G.V.; Mining. CNN Explainer. Available online: <https://poloclub.github.io/cnn-explainer/> (accessed on 19 February 2024).
33. Jetson Nano 2GB Developer Kit-Get Started. Available online: <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-2gb-devkit> (accessed on 19 February 2024).
34. What Is a Key Performance Indicator (KPI)?—KPI.org. Available online: <https://www.kpi.org/kpi-basics/> (accessed on 19 February 2024).

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.