UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

MATHIAS RODRIGUES DA LUZ

PROJETO E DESENVOLVIMENTO DE ASSISTENTE DE VOZ PARA CONTROLE DE PAINEL DE INSTRUMENTOS AUTOMOTIVO

THESIS

PONTA GROSSA

2022

MATHIAS RODRIGUES DA LUZ

DESIGN AND DEVELOPMENT OF A VOICE ASSISTANT FOR AUTOMOTIVE DASHBOARD CONTROL

Projeto e Desenvolvimento de Assistente de Voz para Controle de Painel de Instrumentos Automotivo

Thesis presented as a requirement to obtain the title of Master in Electrical Engineering, at the Federal University of Technology – Paraná (UTFPR).

Advisor: Prof. Dr. Max Mauro Dias Santos
Co-advisor: Prof. Dr. Rui Tadashi Yoshino

PONTA GROSSA

2022

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Ponta Grossa

MATHIAS RODRIGUES DA LUZ

PROJETO E DESENVOLVIMENTO DE ASSISTENTE DE VOZ PARA CONTROLE DE PAINEL DE INSTRUMENTOS AUTOMOTIVO

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestre Em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Controle E Processamento De Energia.

Data de aprovação: 21 de Dezembro de 2022

Dr. Max Mauro Dias Santos, Doutorado - Universidade Tecnológica Federal do Paraná

Dr. Evandro Leonardo Silva Teixeira, Doutorado - Universidade de Brasília (Unb)

Dra. Fernanda Cristina Correa, Doutorado - Universidade Tecnológica Federal do Paraná

Documento gerado pelo Sistema Acadêmico da UTFPR a partir dos dados da Ata de Defesa em 21/12/2022.

ACKNOWLEDGEMENTS

Agradeço primeiramente a Deus pela minha vida e por ter me dado saúde e forças para finalizar este projeto de pesquisa.

Aos meus pais, Angela e João Carlos, pelo incentivo na educação e por todo amor, carinho, apoio e suporte na minha vida.

À minha família, por todo carinho, amor, compreensão e incentivo na pesquisa.

À minha noiva, Marcela, por todo amor, carinho e cuidado comigo. Por todo apoio, incentivo, compreensão e ajuda principalmente no período de realização deste projeto de pesquisa.

Ao meu orientador, professor Dr. Max Mauro Dias Santos, pelo apoio no desenvolvimento da dissertação, auxílio durante a pesquisa, dedicação, confiança e incentivo.

Aos professores do Programa de Pós-Graduação em Engenharia Elétrica (PPGEE) da Universidade Tecnológica Federal do Paraná (UTFPR) - Câmpus Ponta Grossa e à todos os professores que contribuíram na minha trajetória acadêmica.

Aos membros da banca, professora Dra. Fernanda Cristina Corrêa e professor Dr. Evandro Leonardo Silva Teixeira, por todas as contribuições ao projeto de pesquisa.

Em especial ao Engenheiro Ricardo Sugayama da Renault, por me apresentar este projeto desafiador, pelo suporte e tempo dedicado ao projeto, assim como por sua paciência e por todos os ensinamentos passados.

A todos os membros do Grupo de Sistemas Automotivos (GSA), especialmente ao Gabriel Siqueira e ao Hugo Leal pela ajuda com os testes finais do projeto.

A todos os voluntários que participaram do processo de gravação de comandos de voz para a geração do banco de dados apresentado nesse trabalho.

À Renault pela oportunidade de participar do projeto e ao grupo de afinidade Access@Renault, pela iniciativa da pesquisa e pela confiança depositada.

Ao professor Dr. Ariel Orlei Michaloski pelo empréstimo do decibelímetro e do calibrador acústico para a realização dos testes finais do projeto.

Aos laboratórios Industry 4.0 e Engineering Design Process (EDP), sob responsabilidade do professor Dr. Rui Tadashi Yoshino, pelos materiais disponibilizados, que foram fomentados pela Agência Brasileira de Desenvolvimento Industrial (ABDI).

Enfim, às pessoas com quem convivi ao longo desses anos, que me incentivaram e que de alguma forma contribuíram para a realização deste projeto de pesquisa.

Agradeço também à Fundação Araucária e à Fundação de Apoio à Educação, Pesquisa e Desenvolvimento Científico e Tecnológico da Universidade Tecnológica Federal do Paraná (FUNTEF-PR), pelo apoio financeiro em parceria com a Renault do Brasil, por meio do edital 21/2021 - PROPPG (reabertura do edital para seleção de bolsistas em vagas remanescentes).

RESUMO

LUZ, Mathias Rodrigues da. **Projeto e Desenvolvimento de Assistente de Voz para Controle de Painel de Instrumentos Automotivo**. 2022. 133 f. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2022.

Para obter a carteira nacional de habilitação, os motoristas brasileiros passam por exames de aptidão física e mental, de modo a comprovar que são capacitados para operar um veículo. Entretanto, as pessoas que não estão em plenas capacidades físicas, denominadas Pessoas com Mobilidade Reduzida, não possuem a mesma liberdade para se locomover devido as suas limitações. Eles enfrentam dificuldades com a mobilidade urbana, ficando dependentes do transporte público ou por aplicativos. Outra alternativa é o uso de veículos especialmente adaptados, porém até mesmo esses veículos não suprem todas as suas necessidades. Com o advento de novas tecnologias de reconhecimento de voz e de sistemas avançados de assistência ao motorista, surgiram sistemas que permitem controlar o veículo por comandos de voz, que atuam em funções multimídia ou controle climático, por exemplo. Sendo assim, este trabalho propõe desenvolver um sistema embarcado para auxiliar o motorista na condução do veículo usando reconhecimento de fala e foca em criar um protótipo para avaliar a usabilidade do sistema por uma Prova de Conceito. O modelo V de desenvolvimento de software foi utilizado como base da metodologia para a criação de um assistente de voz capaz de reconhecer quatro comandos (seta para direita, seta para esquerda, pisca alerta e luz baixa) e controlar as respectivas funções do veículo. O reconhecimento de comandos de voz foi feito utilizando uma verificação por três etapas com a aplicação de técnicas de inteligência artificial como redes neurais e aprendizado profundo. Este trabalho também descreve a criação de um banco de dados com comandos de voz em Português Brasileiro para treinamento de modelos de reconhecimento de fala, por meio de Transferência de Aprendizado. Além de reconhecer os comandos de voz, o assistente é capaz de identificar o motorista pela voz e verificar a similaridade dos comandos com a voz do motorista. O sistema desenvolvido atendeu a todos os requisitos estabelecidos na etapa de projeto, e reconheceu corretamente 98.4% dos casos explorados, sem ruídos. Nos demais casos, nenhum comando foi reconhecido, o que é considerado melhor que o reconhecimento de outro comando, visto que isto acarretaria na atuação de uma função indesejada. Além disso, o protótipo desenvolvido foi testado em um veículo em seis situações reais de direção, com o ruído sonoro sendo monitorado. O sistema funcionou perfeitamente até uma média de 73.8 dB, que corresponde ao nível sonoro característico dentro de veículos em movimento. O tempo de processamento dos comandos de voz foi de aproximadamente 1 segundo.

Palavras-chave: assistente de voz; sistemas embarcados; modelo v; reconhecimento de fala; assistência ao motorista.

ABSTRACT

LUZ, Mathias Rodrigues da. **Design and Development of a Voice Assistant for Automotive Dashboard Control**. 2022. 133 p. Thesis (Master's Degree in Electrical Engineering) – Federal University of Technology – Paraná. Ponta Grossa, 2022.

To obtain a national driver's license, Brazilian drivers undergo physical and mental aptitude tests to prove they are qualified to operate a vehicle. However, people who are not in full physical capacity, called Persons with Reduced Mobility, do not have the same freedom to move around due to their limitations. They face difficulties with urban mobility, becoming dependent on public transportation or ride-hailing apps. Another alternative is using specially adapted vehicles, but even these vehicles do not meet all their needs. With the advent of new speech recognition technologies and advanced driver assistance systems, new systems that control the vehicle by voice commands have emerged, acting on multimedia functions or climate control, for example. Thus, this work proposes to develop an embedded system to assist the driver in vehicle conduction using speech recognition and focuses on creating a prototype to evaluate the system's usability by a Proof of Concept. The V-model of software development was used as the basis of the methodology to create a voice assistant capable of recognizing four commands (right turn signal, left turn signal, hazard warning, and headlights) and controlling the respective functions of the vehicle. The recognition of voice commands was done using a three-step verification that applied artificial intelligence techniques such as neural networks and deep learning. This work also describes the creation of a database of Brazilian Portuguese voice commands for training speech recognition models through Transfer Learning. Besides recognizing the voice commands, the assistant can identify the driver by voice and verify the similarity of the voice command with the driver's voice. The developed system met all the requirements established in the design stage and correctly recognized 98.4% of the explored cases without noise. In the other cases, no commands were recognized, which is considered better than recognizing another command since this would result in the actuation of an undesired function. Furthermore, the developed prototype was tested in a vehicle in six real driving scenarios, with the sound noise being monitored. The system worked perfectly up to an average of 73.8 dB, which corresponds to the characteristic sound level inside moving vehicles. The processing time for the voice commands was approximately 1 second.

Keywords: voice assistant; embedded systems; v-model; speech recognition; driver assistance.

LIST OF ALGORITHMS

LIST OF FIGURES

LIST OF PHOTOGRAPHS

LIST OF GRAPHS

LIST OF FRAMES

LIST OF TABLES

LIST OF ACRONYMS

Initialism	
ADAS	Advanced Driver Assistance Systems
AI	Artificial Intelligence
AM	Acoustic Model
ANN	Artificial Neural Network
API	Application Programming Interface
ASR	Automatic Speech Recognition
AWS	Amazon Web Services
BP	Brazilian Portuguese
BPTT	Backpropagation Through Time
CER	Character Error Rate
CNN	Convolutional Neural Network
CSLU	Center for Spoken Language Understanding
CTELL	Center for Technology Enhanced Language Learning
DFL	Department of Foreign Languages
DL	Deep Learning
DNN	Deep Neural Network
DSP	Digital Signal Processing
EER	Equal Error Rate
ELRA	European Language Resources Association
FN	False Negatives
FP	False Positives
FRR	False Rejection Rate
GE2E	Generalized end-to-end
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
HMM	Hidden Markov Model
HTK	Hidden Markov Model Toolkit
IoT	Internet of Things
KIT	Karlsruhe Institute of Technology
KWS	Keyword Spotting
LCN	Locally Connected Deep Neural Network
LED	Light-Emitting Diode
LM	Language Model

MER	Match Error Rate
ML	Machine Learning
MLS	Multilingual LibriSpeech
MR	Misclassification Rate
mTEDx	Multilingual TEDx
MVP	Minimum Viable Product
PLDA	Probabilistic Linear Discriminant Analysis
PoC	Proof of Concept
PPGEE	Programa de Pós-Graduação em Engenharia Elétrica
PRM	Persons with Reduced Mobility
RNN	Recurrent Neural Network
SDLC	Software Development Life Cycle
STT	Speech-to-Text
TD-SV	Text-Dependent Speaker Verification
TI-SV	Text-Independent Speaker Verification
TN	True Negatives
TOR	Take-Over Request
TP	True Positives
TTS	Text-to-Speech
UTFPR	Universidade Tecnológica Federal do Paraná
UX	User Experience
VA	Voice Assistants
VAD	Voice Activity Detection
VUI	Voice User Interface
VVCD	Vehicle Voice Commands Dataset
VVCMD	Vehicle Voice Commands Mixed Dataset
WER	Word Error Rate
WIL	Word Information Lost
WIP	Word Information Preserved

CONTENTS

1 INTRODUCTION

Persons with Reduced Mobility (PRM) experience difficulties circulating in Brazilian cities due to poor infrastructure, inadequate sidewalks, and a lack of access ramps (OLIVEIRA *et al.*, 2012; Mobilize Brasil, 2019). As a result, they frequently need to rely on public transportation or ride-hailing apps, which are often unprepared to meet their needs (IBGE, 2021; LISBOA, 2021).

For this reason, some decide to drive vehicles specially adapted for their condition. However, these adaptations do not meet all their needs, and they sometimes need to take their hands off the adapted control for manual acceleration and braking to operate the turn signal lever, for example, which is a risk to their lives and traffic in general.

Voice Assistants (VA) allow hands-free control of devices. Most are embedded in smartphones or dedicated home smart speakers (HOY, 2018). Their applications include controlling media playback, Internet of Things (IoT) devices like lights and alarms, checking for calendar events, making phone calls, and even sending and reading email and text messages (HOY, 2018). Most recently, even accessibility applications have been addressed (PRADHAN *et al.*, 2018; MASINA *et al.*, 2020; BRUNETE *et al.*, 2021). Nowadays, VAs applications include a variety of purposes, like healthcare information, education, shopping, social interaction and entertainment, and even for automotive vehicles.

Although there are already voice assistants in commercial vehicles in Brazil, most are only for controlling multimedia, and none aims to activate dashboard functions such as headlights and turn signals. It is possible to help drivers with reduced mobility by allowing them to control vehicle functions by voice.

1.1 Proposition

That way, we propose developing a system capable of activating vehicle functions by voice commands in Brazilian Portuguese (BP). These functions include but are not limited to headlights and turn signals control, which can help drivers of adapted vehicles.

1.2 Objectives

The main objective of this work is to assess the usability of a voice assistant to control dashboard functions of an adapted vehicle for PRM.

The project targets four specific objectives to accomplish this goal:

- Define User Requirements;
- Create a database of voice commands;
- Develop an application capable of interpreting the voice commands coming from the driver;
- Validate the designed system on a Proof of Concept (PoC).

1.3 Motivation

The motivation for the work came from a demand from the Access@Renault group, which aims to develop inclusion projects for people with disabilities and improve their quality of life. They requested a solution to activate some functions of the vehicle's dashboard through voice commands. That way, drivers with reduced mobility would be able to operate these commands hands-free, which is essential as they may be busy controlling other vehicle functions, such as braking and acceleration, depending on the type of vehicle adaptation, which varies from driver to driver based on their disability.

1.4 Relevance

Despite being among the ten most spoken languages in the world, Portuguese has far fewer Automatic Speech Recognition (ASR) resources than English, Mandarin, and Spanish (LIMA; COSTA-ABREU, 2020). That way, it is considered an under-resourced language, with great potential for research and development of new resources.

Furthermore, most developed ASR systems run on servers, which makes them dependent on connectivity (FENDJI *et al.*, 2022). Therefore, research on new embedded ASR systems is crucial for in-vehicle voice assistance in Brazil since automotive connectivity is a still-growing technology in the country.

1.5 Delimitation

This work focuses on the design and development of a small footprint system for on-device voice commands recognition.

In this way, the voice assistant should interpret commands and execute them as quickly as possible, ensuring the assertiveness of the commands and the safety of the passengers in the vehicle while also providing feedback to the driver regarding the understanding of the commands, but not being an assistant that must understand everything and talk in natural language.

In addition, the study seeks to test two hypotheses:

- H1 - Speech similarity can be used to estimate voice commands;
- H2 - Spectrogram classification based on Convolutional Neural Network (CNN) can be used to enhance command prediction.

1.6 Contributions

One of the main contributions of this work is the creation of a new BP dataset consisting of voice commands for controlling vehicle dashboard functions. Google Colaboratory was used to develop an application called Audio Dataset Creator, to enable participants to record speeches anywhere with any device connected to the internet.

The application instructed the participants on how they should interact with it. Google Speech-to-Text (STT) was used to obtain the transcript of each audio, to validate through comparison with the target command prompt. It is also a contribution of this work, since it can be easily adapted to create other audio databases and is publicly available.

The main contribution of this dissertation is the development of a new vehicle voice assistant to control dashboard functions through voice commands in Brazilian Portuguese, which can help drivers with reduced mobility in vehicle conduction. We have also created a user manual with English and Portuguese versions.

1.7 Thesis structure

This chapter presented the problem statement and the motivation for this work. Additionally, it states the proposition, the objectives, the relevance, the delimitation, and the contributions

of the research.

The remainder of this master thesis consists in four chapters, which are as follows:

The main topics related to the development of the proposed system are presented in Chapter 2 as a background to figure out the core of this project.

In Chapter 3, the system design process is presented, followed by the methodology used for the system development.

Chapter 4 presents the results from the tests of the developed functions and the logs from the deep learning models training. This chapter also includes a discussion of these findings.

Finally, Chapter 5 presents the conclusions and describes some future works that can be made to improve the project.

2 LITERATURE REVIEW

In this Chapter, we will present the main topics related to this project, which are necessary for understanding this work. In addition, a literature review will be presented on works related to the topics:

- Vehicle Voice Assistants;
- Automatic Speech Recognition;
- Brazilian Portuguese Voice Datasets;
- Speaker Recognition.

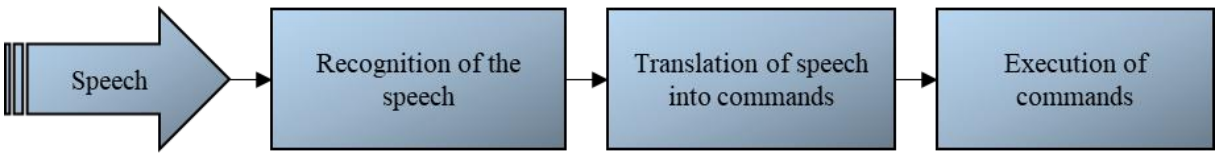
2.1 Theoretical background

According to the latest National Health Survey, conducted in 2019, 8.4% of the Brazilians over the age of 2 have some kind of disability (FIOCRUZ, 2019). People who have motor disabilities, in the lower or upper limbs, correspond to 4.9% of the country's population. They face difficulties with urban mobility due to poor accessibility in Brazilian cities, with inappropriate sidewalks and lack of access ramps (OLIVEIRA *et al.*, 2012; Mobilize Brasil, 2019). As a result, they frequently need to rely on public transportation or ride-hailing apps, which are often unprepared to meet their needs (IBGE, 2021; LISBOA, 2021).

Some choose to drive a specially adapted vehicle for their condition. However, these adaptations do not meet all their needs, and a voice assistant capable of activating vehicle functions would help them to drive more safely.

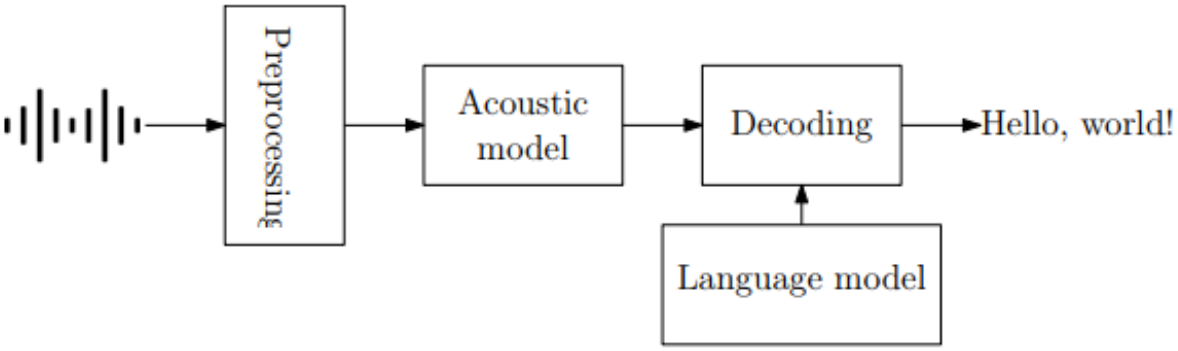
An implementation of a voice controlling system is presented in Figure 1. It starts with the speech recognition, which can be performed by an automatic speech recognition system, as the presented in Figure 2.

Figure 1 – Steps of a voice control system.



Source: Matarneh *et al.* (2017).

Figure 2 – Automatic speech recognition system.



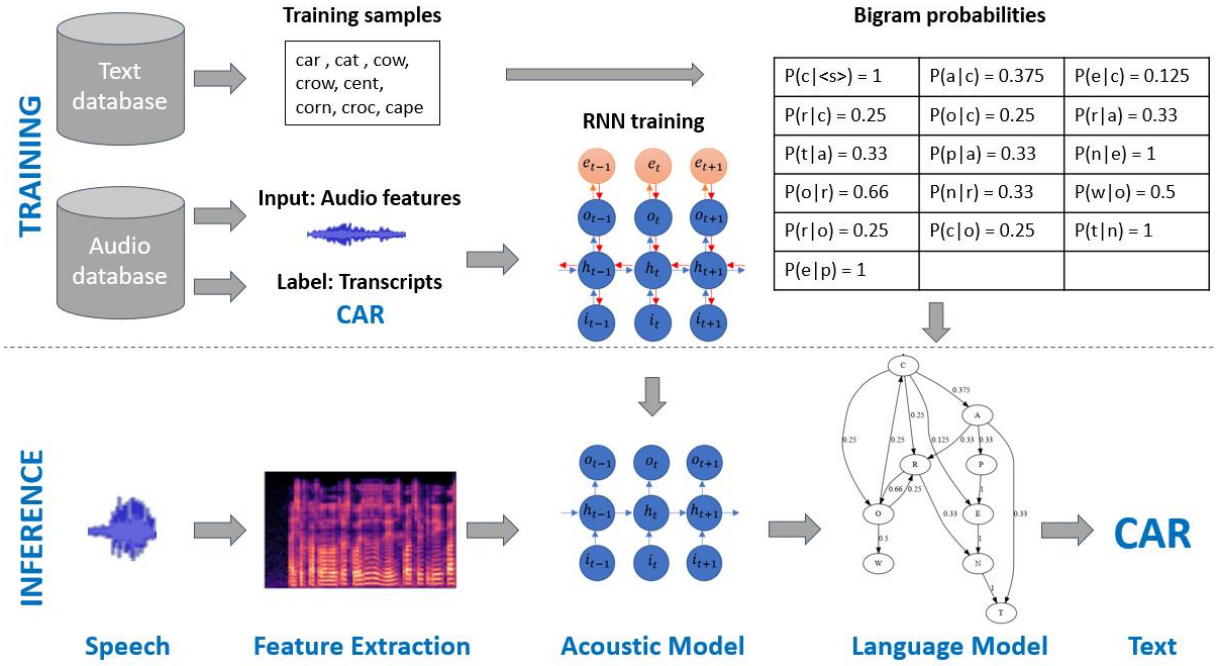
Source: Quintanilha *et al.* (2020).

An ASR system converts audio to text using an Acoustic Model (AM) and a Language Model (LM). Then, the obtained text can be used for translating the speech into a voice command.

An acoustic model converts audio to written text based on the probability of characters in the alphabet. It predicts which phoneme corresponds to each waveform and is used to recognize speech. The language model, in turn, is used to improve the transcription's accuracy by calculating the probability of a sequence of words happening.

Figure 3 illustrates the training process of the acoustic and language models and their usage for inference on new data.

Figure 3 – Speech-to-Text training and inference.

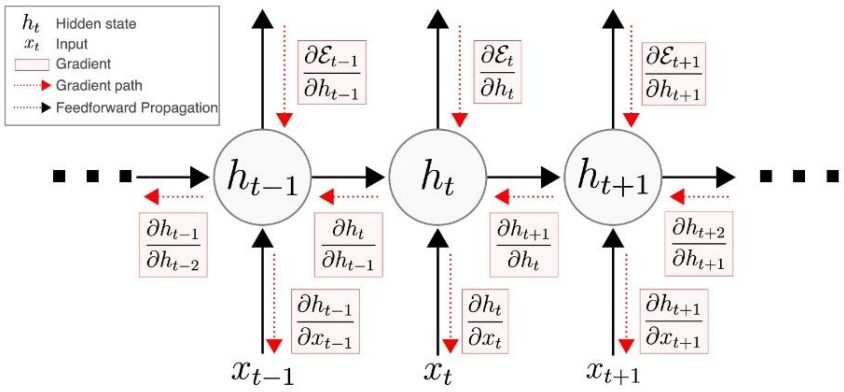


Source: Own authorship (2022).

The AM is created by training a Recurrent Neural Network (RNN), which consists of a Deep Learning (DL) architecture highly used for natural language processing and other cases where sequences are essencial. RNN models create an internal memory, which stores temporal information and can handle temporal sequences like videos or sentences.

The RNN training can start with random weights to calculate the output used to compute the error, which serves to update the weights via Backpropagation Through Time (BPTT). Figure 4 exemplifies the RNN training, including the process of BPTT represented by the gradients and gradient paths (red arrows).

Figure 4 – Backpropagation Through Time (BPTT) for Recurrent Neural Network (RNN) training.



Source: Adapted from Lillicrap and Santoro (2019).

The language model calculates the probability $P(w)$ of a sequence of words w (w_1,w_2,...,w_N) to happen (FENDJI *et al.*, 2022), through Equation 1.

$$P(w) = \prod_{i=1}^{N} P(w_i|w_1,...,w_{i-1}) = \prod_{i=1}^{N} P(w_i|h_i), \tag{1}$$

where $h_i = w_1,...,w_{i-1}$ is the history of the word w_i and $P(w_i|h_i)$ is the probability of the word w_i knowing all the previous ones.

The n-gram approach is used to reduce the LM complexity and consists in limiting the history size to $n-1$ words. That way, the probability of the sequence w of words is calculated by:

$$P(w) = \prod_{i=1}^{N} P(w_i|w_{i-n+1},...,w_{i-1}) \tag{2}$$

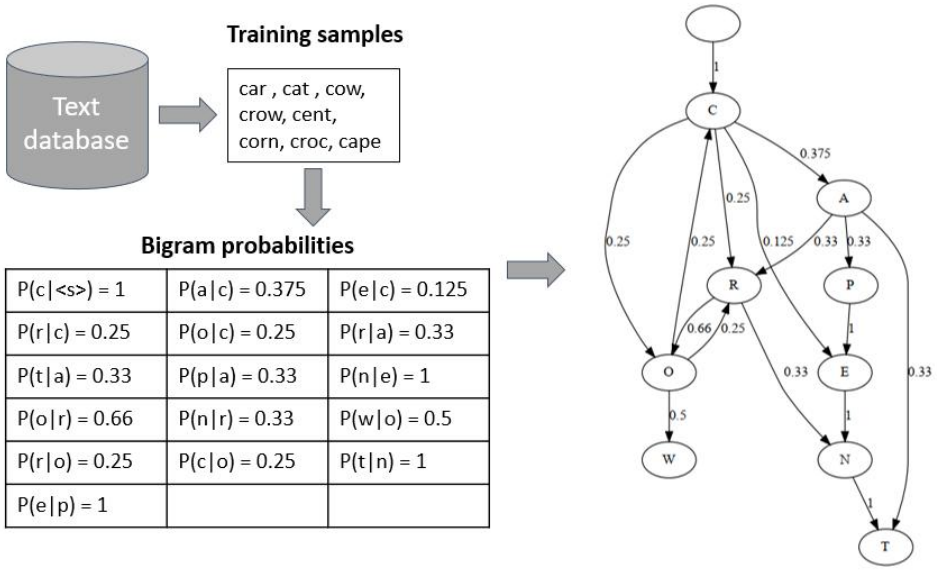
The probability of each word w_i is calculated by:

$$P(w_i) = \frac{C(w_i)}{C}, \tag{3}$$

where $C(w_i)$ is the number of occurrences of the word w_i and C is the total number of words in the training corpus.

Figure 5 presents an example of a bigram, an n-gram with n equals to two. In this case, it is at the character level, which means that it calculates the probability of a character to appear, given the last one.

Figure 5 – Example of a bigram language model with its calculated probabilities.



P(c|<s>) = 1	P(a|c) = 0.375	P(e|c) = 0.125
P(r|c) = 0.25	P(o|c) = 0.25	P(r|a) = 0.33
P(t|a) = 0.33	P(p|a) = 0.33	P(n|e) = 1
P(o|r) = 0.66	P(n|r) = 0.33	P(w|o) = 0.5
P(r|o) = 0.25	P(c|o) = 0.25	P(t|n) = 1
P(e|p) = 1		

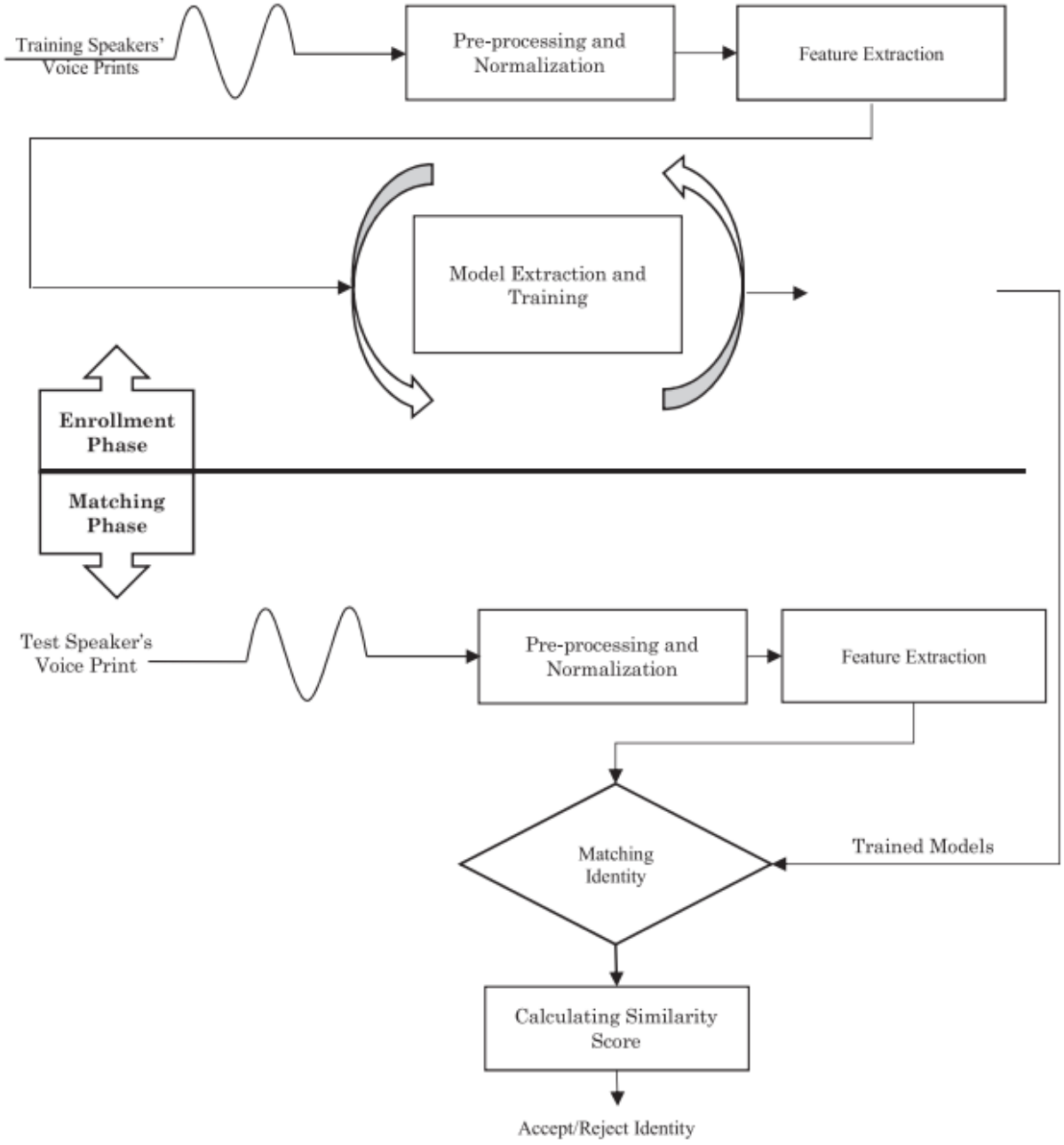
Source: Own authorship (2022).

With this language model, the probability of the word "car", for example, would be calculated by Equation 4.

$$P(car) = P(c| < s >) * P(a|c) * P(r|a) = 1 * 0.375 * 0.33 = 0.125 \tag{4}$$

For systems like the proposed, speaker recognition is crucial, as only the driver should be able to control vehicle functions. The phases of speaker recognition systems are presented in Figure 6.

Figure 6 – Phases of speaker recognition systems.



Source: Tirumala *et al.* (2017).

The first phase is the enrollment, in which the speaker model is created, usually by averaging the speaker representation vectors with frame-level information (HEIGOLD *et al.*, 2016). One way to obtain speaker representation vectors is by using a Deep Neural Network (DNN), which is a type of Artificial Neural Network (ANN) with multiple hidden layers and is also known as DL. ANN is a field of artificial intelligence that attempts to replicate the capabilities of biological neural networks related to decision-making and learning. The first studies in the field are from McCulloch and Pitts (1943), with the proposal of a mathematical

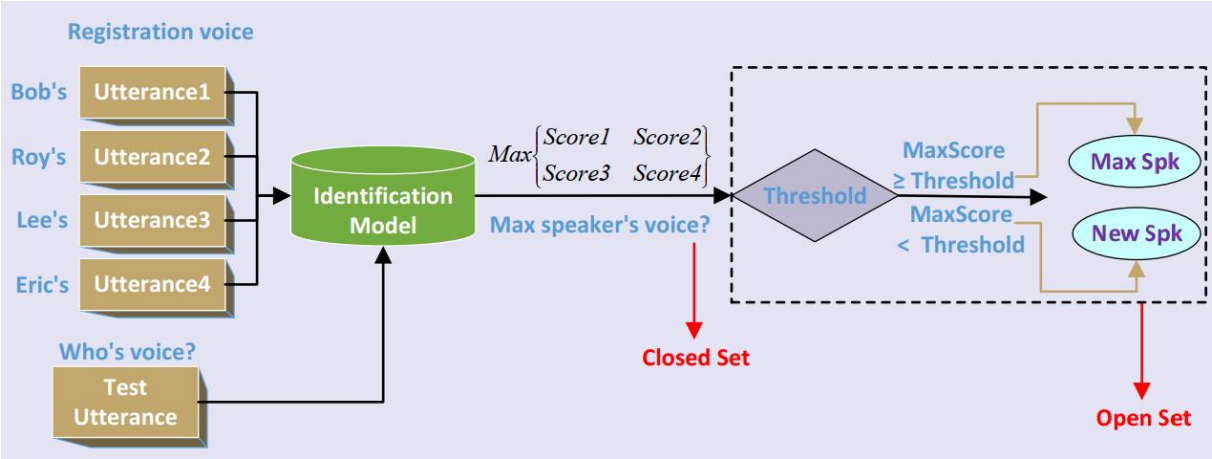
neuron model. ANNs are the basis for Machine Learning (ML).

The second phase is the identity matching, in which the obtained models are compared to the tested utterance by using the cosine similarity between the voice features embedding vectors. The cosine similarity is calculated by Equation 5.

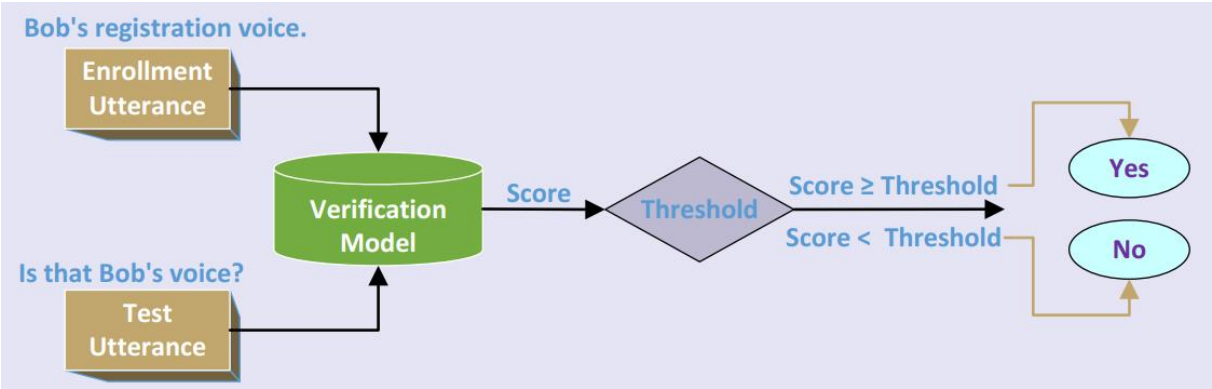
$$cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\left\|\mathbf{A}\right\| \left\|\mathbf{B}\right\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}, \tag{5}$$

where A and B represent two voice embedding vectors, and θ is the angle between the two vectors. For the matching phase, there are two main types of speaker recognition systems, which are presented in Figure 7: identification and verification.

Figure 7 – Speaker identification and verification processes



(a) Speaker identification process



(b) Speaker verification process

Source: Adapted from Bai and Zhang (2020).

In speaker identification, presented in Figure 7(a), the system tries to recognize the user by its voice. It determines which speaker originated a given utterance by comparing the voice biometrics with stored speaker models (NAGRANI *et al.*, 2017). The voice similarity is

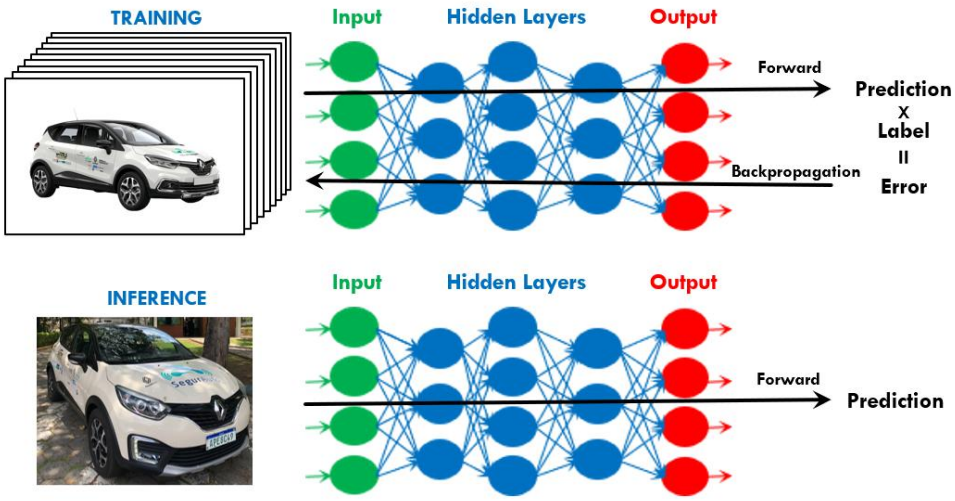
compared to all stored voice models, obtained during the enrollment phase. The speaker model that has the maximum similarity score is considered the identified speaker if the score is higher than a threshold.

Instead of determining who spoke, a speaker verification system (shown in Figure 7(b)) accepts or rejects the identity claimed by the speaker. It can be classified as Text-Dependent Speaker Verification (TD-SV) and Text-Independent Speaker Verification (TI-SV). The first refers to when the transcripts of reference enrollment and verification utterances are constrained to a specific sentence, and TI-SV refers to when there are no constraints (WAN *et al.*, 2017).

A CNN is another architecture of DL used to extract features and classify the input data. It is usually used for image classification, which is the task of categorizing images by assigning labels to them. For this, a previously trained CNN model is used.

The training of an image classification model is a supervised learning problem, with a label assigned to each image in the training dataset. It can start with random weights to calculate the model's output. The training is responsible for updating the weights through the backpropagation of the gradients of the errors calculated by comparing the label supplied with the output. Figure 8 illustrates the training process of a CNN model and the inference process using the trained model.

Figure 8 – Convolutional Neural Network (CNN) for image classification: training and inference.

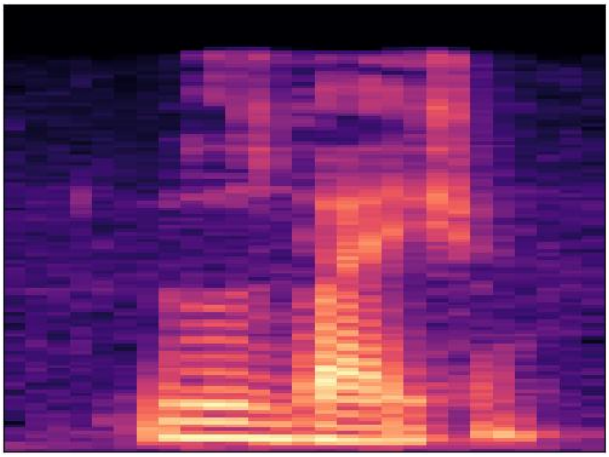


Source: Own authorship (2022).

It can be used to classify spectrograms of the recorded voice with an image classification model. A spectrogram can be described as a picture of the sound. It is an image representation of the signal's frequency spectrum, with time on the horizontal axis, frequency on the vertical

axis, and colors representing the strength of a frequency component at each time frame (WYSE, 2017). That way, spectrograms may help classify the sound and identify spoken words. Figure 9 represents a spectrogram generated by a voice command.

Figure 9 – Spectrogram of a voice command.



Source: Own authorship (2022).

2.2 Vehicle voice assistants

Voice assistants have grown in popularity, especially since smart speakers incorporated them. Their applications range from playing music and games to controlling other connected devices (TERZOPOULOS; SATRATZEMI, 2020).

Alvarez *et al.* (2010) presented a Voice User Interface (VUI) capable of answering user questions about the vehicle's functions. A database was designed with a list of possible questions for each vehicle selected feature, mapped to page numbers from the 2010 BMW 7 Series manual. They used the Answer First methodology to allow users to speak naturally. Each question from the list, and the user question, are decomposed into bigrams, a sequence of two consecutive words, to determine the questions with the highest concentration of matched bigrams. Then, the system answers the user with the most probable page number from the manual that explains the requested feature.

A conversational in-vehicle digital assistant named Adasa was developed in 2018 to help drivers understand Advanced Driver Assistance Systems (ADAS) features (LIN *et al.*, 2018). They trained the assistant using ML techniques to respond to drivers' questions and commands in natural language. According to the authors, Adasa captures the driver's voice with a microphone after being activated by the driver through the voice activation button on the steering wheel.

Then, the audio transcript, obtained through STT, is classified to determine whether the system should get information from the vehicle or the knowledge base. Finally, the response is output to the vehicle speaker through Text-to-Speech (TTS) to answer the driver's questions.

In 2019, the use of personalized voice assistants to improve User Experience (UX) was investigated by Braun *et al.* (2019). Four different personalities were used in the assistant and compared to a default assistant. They conducted a pre-study with 19 participants to determine the most appropriate personalities for an in-car assistant. They created a decision tree to assign 55 participants to one of the four assistants based on their personalities. They opted for a Wizard-of-Oz methodology (DAHLBäCK *et al.*, 1993) with an operator (wizard) in the vehicle's back seat while each participant was driving. Twelve use cases, divided into three clusters (Driving Related, Proactive Assistant, Connected Car), were triggered by the operator along the way. Each participant experienced two rides, with the default assistant and the personalized previously assigned, and answered questionnaires after each ride.

According to the results presented by Braun *et al.* (2019), a personalized vehicle voice assistant is more suitable for non-driving-related situations but not as appropriate for security-relevant driving situations. The UX of participants who experienced a correctly matched personalized assistant improved compared to the default. But users preferred the default when the personalized VA was not correctly assigned, suggesting that the driver should be able to select the assistant personality for each case.

Schmidt *et al.* (2020) evaluated user acceptance of Proactive Assistant. They conducted a Wizard-of-Oz study consisting of six use cases in a driving simulator with 42 participants. Each participant had uninformed interactions with both proactive and non-proactive VA. The traffic simulations had four variants, alternating between proactive and non-proactive and changing traffic density. The use cases covered by the study were news, parking, break, refueling, rerouting, and appointment, with the last 2 having the highest acceptance of the proactive assistant.

Shah and Gusikhin (2020) described an experimental system to manage vehicle climate control using Amazon Alexa VA and SmartDeviceLink Application Programming Interface (API). They also controlled a scent dispenser. They developed an Alexa Custom Skill connected to the Amazon Web Services (AWS) to implement dialogue to manage vehicle ambiance. They suggested eliminating the invocation word and adding more features like radio and ambient light control as future enhancements to the dialogue system.

Mahajan *et al.* (2021) conducted a study using a driving simulator capable of simulating

a level 3 autonomous vehicle to assess the effectiveness of a VA as a countermeasure to the fatigue effects of autonomous driving. A total of 24 drivers participated. They took two drives, with and without VA, in the same scenario, whose route was about 30 minutes, with 25 minutes of automated driving followed by a Take-Over Request (TOR) from the vehicle. The authors developed a VA that could provide information about weather, traffic, calendar event reminders, and more to keep the driver engaged. The results showed that a VA could improve drivers' alertness during automated driving.

2.3 Automatic speech recognition

ASR is crucial to the proposed system's development because it can be considered the first step in a voice assistant pipeline. It is employed to convert audio to text.

Lei *et al.* (2013) described the development of a speech recognition system for large vocabulary dictation on mobile devices. Their objective was to develop a fast and accurate, not internet-dependent, speech recognition system to run directly on the device, since most ASR systems run on servers. Because of this, they focused on creating a small-footprint AM based on DNN, which proved to be an improvement to the previously used Gaussian Mixture Model (GMM). They achieved a Word Error Rate (WER) of 15.1% with an AM containing 2.7M parameters along with a compressed LM (3.7MB file size), while the GMM model got 20.7% WER with 8.08M parameters (14MB file size) and the server DNN got 12.3% WER with 49.3M parameters (50.8MB file size).

Deep Speech, a speech recognition system created using end-to-end deep learning, is presented by Hannun *et al.* (2014). They trained a RNN to generate English text transcriptions from speech spectrograms. A dataset of 5000 hours of reading speech from 9600 speakers was used to train one of their models. They used the superposition of signals to generate noisy training data, aiming to improve speech recognition in noisy environments. The model trained with the 5000-hour dataset with noise addition got a WER of 22.6% on 100 noisy audios from the test subset, while the model trained with the same dataset without noise addition got a WER of 28.7%. They also compared a Deep Speech model, trained with more than 7000 hours with synthesized noise, to four commercial speech systems: wit.ai, Google Speech API, Bing Speech, and Apple Dictation, with their model standing out from the others in 82 noisy and 94 clean test audios.

By the year 2015, DNN models were used for Keyword Spotting (KWS) since they

outperformed Hidden Markov Model (HMM) systems, which were commonly used in this task (SAINATH; PARADA, 2015). However, Sainath and Parada (2015) proposed a CNN architecture for KWS because of the benefits of performance and reduced model size of CNN over DNN. They selected 14 keyword phrases containing two words, such as "play music" and "answer call", to serve as the dataset for training the models and further comparison. They showed that the best performing CNN has improved over 41% relative to DNN in clean and noisy audios and that the best performance of CNN is achieved using a pooling technique.

Warden (2018) described the creation of a command dataset for speech recognition. It focused on a limited English vocabulary, including the digits from zero to nine and 14 commands for IoT and robotics applications. Besides that, the author added 11 words to test the model's ability to ignore speech that does not contain commands. He chose some because they share similarities to one of the commands and others as they covered a lot of different phonemes. All utterances were captured through phone or laptop microphones, with no Digital Signal Processing (DSP) technique applied to reduce noise to reflect the on-device trigger keyword detection task.

Warden (2018) created an open-source web-based application to record 135 audios per user, with a duration of 1.5 seconds. He decided to record isolated words on each audio since it better resembles the keyword recognition task desired. After the recording, the audios were passed through an audio processing tool to remove silence resulting in audios with a maximum duration of 1 second. Besides that, a manual review was made to assess if the transcript of the audio matches the actual speech. The resulting dataset is composed of 105829 utterances of 2618 speakers. He trained a model with this dataset based on the work of Sainath and Parada (2015) and obtained a Top-One accuracy of 88.2% on the test set.

A survey focused on small vocabulary ASR systems is presented by Fendji *et al.* (2022). They described some ASR principles and approaches, such as ML and DL. They also showed a collection of frameworks and toolkits for speech recognition system development. According to them, the most used toolkit for limited vocabulary (up to 1000 words) ASR systems is the Hidden Markov Model Toolkit (HTK), which is maintained at the Speech Vision and Robotics Group of the Cambridge University Engineering Department (CUED) and is used by 38% of the works selected in the paper. They showed a summary of the recent research on ASR using a limited vocabulary, containing information on the dataset language, the number of speakers, the toolkit or model used, the resulting accuracy, and some other info about the dataset, like the

number of words, sentences, utterances or phonemes, as well as if it has noisy audios or not. They concluded that the papers which used HMM and DNN outperformed the ones that used GMMs as AM.

2.4 Brazilian portuguese voice datasets

OGI 22 (LANDER *et al.*, 1995) is a paid dataset with 22 language corpus, including one in Portuguese, but the authors did not specify the speakers' country of origin. The recordings were made through digital telephone lines, sampled at 8 kHz, and stored in riff format (16 bits - mono channel). The corpus contains fixed vocabulary utterances as well as fluent continuous speech. They received 497 calls from Portuguese native speakers, from which 84 were verified by two other native Portuguese speakers.

Spoltech Brazilian Portuguese (SCHRAMM *et al.*, 2006) is a paid dataset created by the Center for Spoken Language Understanding (CSLU) in association with the Federal University of Rio Grande do Sul (UFRGS) and the University of Caxias do Sul (UCS), which were responsible for recording and transcribing 5 hours of BP speeches. A total of 480 speakers from all over Brazil participated, recording 8207 utterances, 2540 of which were transcribed at the word level without time alignments and 5479 at the phoneme level with time alignment. The utterances are from reading speeches and responses to questions. The audios were recorded in an uncontrolled environment at a sampling rate of 44.1 kHz, mono channel with 16 bits per sample, and saved in riff format. The speaker was responsible for verifying each recording.

West Point (MORGAN *et al.*, 2008) is another paid dataset of BP speech, with approximately 1.6 hours of audio recordings. It was created by the Department of Foreign Languages (DFL) and the Center for Technology Enhanced Language Learning (CTELL) and collected in 1999 in Brasilia (Brazil) with the collaboration of 60 female and 68 male native and non-native speakers who participated by reading a prompt containing 296 sentences typically used in language learning situations. The sample rate of some recordings was 22050 Hz, with 16 bits per sample, and some used 11025 Hz, with 8 bits. The speaker reviewed the audio after each recording. A member of the collection team was also assigned to verify the audios.

CETUC (ALENCAR; ALCAIM, 2008) is a BP speech dataset freely available that was recorded by 50 male and 50 female speakers reading 1000 sentences, with 3528 words, totaling almost 145 hours of audio. The recordings were made in a controlled environment at a sampling rate of 16 kHz with 16 bits per sample.

Sid (QUINTANILHA *et al.*, 2017) is a free BP dataset with audios recorded at 22050 Hz in an uncontrolled environment by 21 females and 51 males. It has 5777 recordings and files with information about each speaker. It also has prompts with the sentences spoken in each audio, whose content ranges from numbers to complex sentences.

GlobalPhone (ASSOCIATION, 2018) is another paid dataset of multilingual speech. It was created by the European Language Resources Association (ELRA) in collaboration with the Karlsruhe Institute of Technology (KIT) and included 22 languages. The BP corpus has 100 sentences from *Folha de São Paulo* newspaper spoken by 54 male and 48 female speakers. The audios were recorded in an office with a Sennheiser 440-6 close-speaking microphone, in 16 bits and a sampling rate of 16 kHz in a mono channel. The dataset has a total of 20 hours of audio.

FalaBrasil (Grupo FalaBrasil, 2020) is a Brazilian research group from the Federal University of Pará (UFPA) focused on speech and natural language processing in Brazilian Portuguese. They are responsible for maintaining 4 BP public datasets:

- *Constituição* has 1255 utterances recorded by a single male speaker reading the constitution of Brazil. The dataset has approximately 9 hours of speech. The recording environment was well controlled.

- *Código de Defesa do Consumidor* has 97 audios, recorded by a single male speaker, reading the consumer protection code from Brazil. The recordings were also made in a controlled environment.

- LaPS Benchmark was recorded by 25 male and 10 female speakers with 20 sentences spoken by each speaker, totaling 700 sentences and 54 minutes of audio. The recordings were made in an uncontrolled environment.

- LaPS Mail consists of 84 minutes of recordings from 21 male and four female speakers. The content of the speeches is a mix of 43 sentences of voice commands to control email and 43 names. The audios were recorded in a non-controlled environment with a high-quality microphone.

All datasets from the FalaBrasil group were recorded with a 16 kHz sampling rate and 16 bits per sample.

Multilingual LibriSpeech (MLS) (PRATAP *et al.*, 2020) is a free multilingual corpus derived from LibriVox audiobooks. It consists of 8 languages: English, German, Dutch, Spanish,

French, Italian, Portuguese, and Polish. It contains resources for ASR and LM training. The Portuguese dataset contains 168.34 hours of audio from 36 male and 26 female voices.

Multilingual TEDx (mTEDx) (SALESKY *et al.*, 2021) is a publicly available dataset of audio recordings from TEDx talks in 8 languages: Spanish, French, Portuguese, Italian, Russian, Greek, Arabic, and German. The Portuguese dataset contains 164 hours of speech with 93000 utterances. The audios were sampled at 44.1 kHz or 48 kHz and stored in wav format.

CORAA (JUNIOR *et al.*, 2021) is a free BP speech dataset. It contains 299.77 hours of audio with its transcriptions, whose content was manually validated to enable ASR and TTS model training. It combines five datasets: ALIP, C-Oral Brasil I, NURC Recife, SP2010, and TEDx Portuguese. Before the validation, the dataset size was 692.21 hours. The total of BP speakers is 1689. At least 65% of the audios are spontaneous speeches.

VoxForge (VOXFORGE, 2022) is a free collection of speech files in 17 languages. According to (QUINTANILHA *et al.*, 2020), it had more than 4 hours of BP speech, recorded by more than 100 speakers in 2020, and may have increased since then. The recordings are from uncontrolled environments, varying the sample rate from 16 kHz to 44.1 kHz.

Tatoeba (TATOEBA, 2022) is a free and open collection of sentences and translations. Trang Ho founded the project in 2006, and it now includes 10660099 sentences in 417 supported languages. It has 405047 sentences in Portuguese, of which 15535 sentences contain audio. It is the 5^{th} language with more audio sentences.

Appen (APPEN, 2022) has 270 paid datasets of audio, image, video, and text in over 80 languages, including 2 BP audio datasets: PTB_ASR001 with 26 hours of scripted speech from 10417 utterances recorded by 102 speakers, and PTB_ASR002 of conversational telephony with 33837 utterances, recorded by 200 participants, totaling 33 hours of recordings.

The Common Voice (v10.0) (MOZILLA, 2022) is a multilingual voice dataset that contains more than 90 languages. It is a free and open-source initiative from Mozilla. The Portuguese dataset contains 2562 voices from people of various ages and genders. The dataset includes 144 hours of speeches, 120 of which were validated.

The Table 1 gives an overview of the 18 voice datasets discussed in this section. Most of the datasets were sampled at 16kHz, with 16 bits per sample. The LaPS Mail stood out, for having voice commands to control email, but none dataset had voice commands to control vehicle functions.

Table 1 – Brazilian Portuguese voice datasets.

Name	Year	Free	Languages	PT Audios	Hours	Participants (M/F)
OGI 22	1995	No	22	84/497	-	497
Spoltech	2006	No	1	807	5	480
West Point	2008	No	1	-	1.6	128 (68/60)
CETUC	2008	Yes	1	100000	145	100 (50/50)
Sid	2017	Yes	1	5777	-	72 (51//21)
GlobalPhone	2018	No	22	-	20	102 (54/48)
Constituição	2020	Yes	1	1255	9	1 (1/0)
Código de Defesa do Consumidor	2020	Yes	1	97	-	1 (1/0)
LaPS Benchmark	2020	Yes	1	700	0.9	35 (25/10)
LaPS Mail	2020	Yes	1	-	1.4	25 (21/4)
MLS	2020	Yes	8	-	168.34	62 (36/26)
mTEDx	2021	Yes	8	93000	164	-
CORAA	2021	Yes	1	-	300/692	1689
VoxForge	2022	Yes	17	-	4+	100+
Tatoeba	2022	Yes	417	15535	-	-
PTB_ASR001	2022	No	1	10417	26	102
PTB_ASR002	2022	No	1	33837	33	200
Common Voice	2022	Yes	90+	-	120/144	2562

Source: Own authorship (2022).

2.5 Speaker recognition

In 2014, Google developers proposed a new approach to obtaining speaker models, using DNN to extract speaker-specific features and averaging these features, also called d-vectors (VARIANI *et al.*, 2014). The input of the DNN is a stack of the log filterbank energy features extracted from an audio frame with its left and right context frames. They showed that increasing the number of utterances used for speaker enrollment improves the speaker verification system. The overall performance of the d-vector is slightly worse than the previously well-established i-vector systems, but they presented a combination of i-vectors and d-vectors that outperformed the Equal Error Rate (EER) obtained by the i-vector system by 14% and 25%, in clean and noisy conditions respectively.

Aiming to create a small footprint TD-SV system to run in real-time in space-constrained mobile platforms, Chen *et al.* (2015) explored alternative architectures to the fully-connected feed-forward DNN architecture used to compute d-vectors by Variani *et al.* (2014). They used Locally Connected Deep Neural Network (LCN) and CNN and reduced 30% of the total model footprint compared to the original size. Additionally, the EER improved by 8% when using the LCN without increasing model size and 10% with the CNN but increasing computation.

Heigold *et al.* (2016) presented an end-to-end TD-SV approach, which compares a pair of speech utterances to compute their similarity. They used a DNN with one locally-connected

layer followed by fully-connected layers to generate utterance d-vectors. To optimize their model architecture, they proposed an end-to-end loss and demonstrated that the utterance-level approach outperforms the frame-level approach by 30%. They also showed that using RNN instead of DNN could reduce EER but at a higher computational runtime cost and that the end-to-end loss achieved the same or even better results than other losses.

Lukic *et al.* (2016) studied the use of CNN for speaker identification and clustering. The used CNN architecture consists of 8 layers and uses mel-spectrograms as input. In terms of identification performance, they achieved an accuracy of 97%, with only 19 misidentified speakers, on the TIMIT Acoustic-Phonetic Continuous Speech Corpus, a dataset with ten sentences read by 630 speakers. In terms of clustering, their best result was a Misclassification Rate (MR) of 0.05 achieved when clustering the outputs of a dense layer, which contains a general representation of the speaker, before the softmax, which labels the speakers.

A CNN architecture to recognize speakers through spectrograms was developed by Nagrani *et al.* (2017). It was based on VGG-M (CHATFIELD *et al.*, 2014) for being good at image classification. They also developed a fully automated pipeline to create a dataset from YouTube videos (VoxCeleb), using active speaker verification with Sync-Net Chung and Zisserman (2016) and face verification with a classification network based on VGG-16 (SIMONYAN; ZISSERMAN, 2014). The resulting dataset is composed of 145000 utterances from 1251 speakers. They demonstrated that their CNN outperformed the current state-of-the-art of speaker identification and verification.

A feed-forward DNN was used for extracting embeddings to replace i-vectors for a TI-SV task (SNYDER *et al.*, 2017). Its foundation is the end-to-end system described by Snyder *et al.* (2016),which offers a framework similar to that of Heigold *et al.* (2016) but aims for text-independent verification. They modified their previous work (SNYDER *et al.*, 2016) to improve performance on smaller datasets by replacing the end-to-end loss with a multi-class cross entropy loss in the DNN that computes speaker embeddings. In addition, they trained a separated Probabilistic Linear Discriminant Analysis (PLDA) for comparing pairs of embeddings. They indicated that embeddings are more effective than i-vectors for short-duration audios while fusing them improves the results.

Generalized end-to-end (GE2E) loss, a new loss function for training speaker verification models, was proposed by Wan *et al.* (2017). They focus on TI-SV and global password TD-SV. Their model builds a similarity matrix with the similarity of each embedding (d-vector) to all

speaker centroids, which is the voice print of a speaker built from the enrollment utterances. They presented an EER improvement of more than 10% compared to their previous work (HEIGOLD *et al.*, 2016) and a model trained with softmax. The training of a model with GE2E was three times faster than with other loss functions.

Snyder *et al.* (2018) improved their previous work (SNYDER *et al.*, 2017) by applying data augmentation to the DNN training process. The embeddings generated by the DNN are called x-vectors, whose system was made available in the Kaldi Toolkit for speech recognition (DANIEL *et al.*, 2011). The x-vectors system outperformed the two systems based on i-vectors on two tested databases.

In 2019, Corentin Jemine (JEMINE, 2019) adapted the framework of real-time voice cloning presented by Jia *et al.* (2018) and made available an open-source application of it. The application uses three stages: a speaker encoder that extracts voice features (embedding), a synthesizer that generates spectrograms from texts based on the speaker embeddings, and a vocoder that creates an audio waveform from the spectrogram. The voice encoder was developed as a side project called Resemblyzer and is based on the work of Wan *et al.* (2017). This master thesis uses Resemblyzer as the voice encoder for extracting audio features, as it is an open-source solution.

3 METHODOLOGY

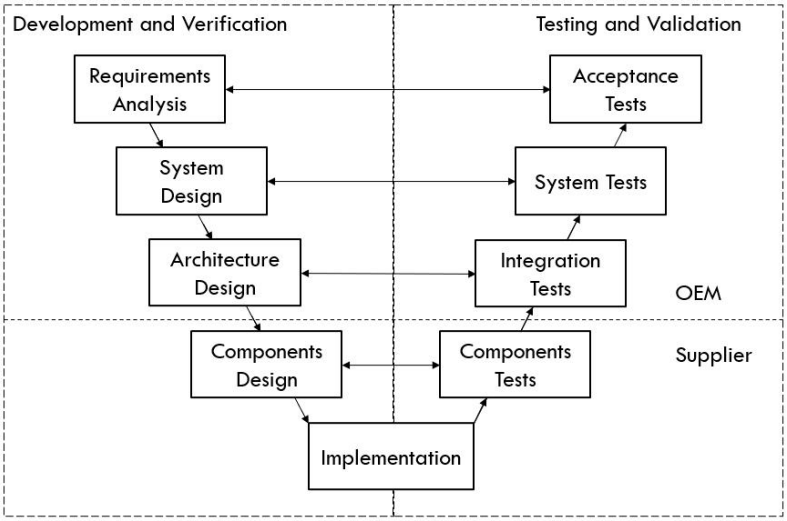
This work aims to develop a voice assistant to control dashboard functions of an adapted vehicle for PRM. In this Chapter, we will present the methodology used for designing and developing the proposed system.

3.1 System development

The system development methodology was based on the V-shaped model of Software Development Life Cycle (SDLC), proposed by Rook (1986).

Figure 10 depicts the system development methodology, with the left side displaying the development and verification steps, which will be described in the following sections of this Chapter, and the right side showing the testing and validation steps that will be presented in Chapter 4.

Figure 10 – V-model workflow for automotive software development.



Source: Own authorship (2022).

3.1.1 Requirements analysis

The development of the VA system started with the definition of the high level requirements, according to the user needs. These requirements are presented in Frame 1.

Frame 1 – System requirements.

Requirement ID	Requirement
R01	The system must start receiving voice commands upon request
R01.1	The system must have a microphone, which is specified in Subsection 3.1.4.1
R01.2	The system must have a button, which is specified in Subsection 3.1.4.1
R01.3	The system must record audio only when required
R02	The system must indicate when it is ready to receive the voice commands
R02.1	The system must have speakers, which are specified in Subsection 3.1.4.1
R02.2	The system must be able to speak and emit sounds warnings
R02.3	The system must instruct the user on how to use the system
R03	The system must be able to verify if the voice command is from the current driver
R03.1	The system must verify if the recorded audio has voice
R03.1.1	The system must report when no speech is detected
R03.2	The system must identify the driver through its voice
R03.2.1	The system must report when the driver is not identified
R03.2.2	The system must report when it identifies the driver
R03.4	The system must store data from new driver enrollments
R03.4.1	The system must be able to store new data from the driver
R03.3	The system must check if the recorded voice is from the driver
R03.3.1	The system must report when the voice fails the verification
R04	The system must respond to specific voice commands, detailed in Subsection 3.2.1.2
R04.1	The system must recognize the voice commands
R04.1.1	The system must report if a voice command was not recognized
R04.2	The system must not act if the command is not from the driver
R04.3	The system must actuate on the vehicle if the command was recognized
R04.3.1	The system must report when a command was received
R04.3.2	The system must check the current status of the vehicle's dashboard before acting

Source: Own authorship (2022).

The requirements were validated using the Simulink Requirements and Test Manager tools, which results are presented in Subsection 4.1.1. Each requirement was linked to at least one test, distributed in system, integration, and component tests.

3.1.2 System design

Based on the established requirements, the system functions can be defined as well as the user interface elements, like the inputs and outputs of the system. Three different versions of the system were defined. However, this work only aims at the development of the first two. The versions are:

- The first (V1) is a prototype that should run in a personal computer, simulating the vehicle actuation step,
- The second (V2) is a prototype that should run in a computer on a test bench, with the vehicle systems being simulated by a microcontroller,

- The last (V3) is the Minimum Viable Product (MVP), which consists of the system running on a vehicle.

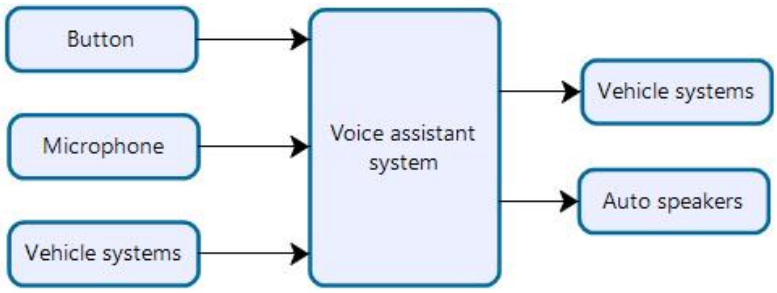
Most ASR systems run on servers (FENDJI *et al.*, 2022), only detecting keywords, also called wake words, on the device and then sending the input audio to be processed in the cloud. Due to connectivity concerns, our system needs to run on-device command recognition. Besides that, a speaker verification step is required to ensure that only voice commands incoming from the driver will be interpreted. There are some constraints related to embedded systems, especially on the model size, which must be small and consume fewer computer resources; moreover, the system must be energy-efficient (WARDEN, 2018).

For command recognition systems, like the proposed one, most of the audio input may be non-speech, being silence or background noise, especially the latter in an automotive environment. That way, false positives must be minimized (WARDEN, 2018). Even when there is speech input, most of it may not be related to the voice commands, as in the case of conversations between passengers or speech over the vehicle's radio. Therefore, arbitrary talks should not trigger the model (WARDEN, 2018).

Figure 11 shows the flow of inputs and outputs of the voice assistant system. The hardware needed consists in:

- A button to trigger the recording, minimizing the concerns on false positives presented by Warden (2018), and to attend requirement R01.2,
- A microphone to record the sound inputs, and to attend requirement R01.1,
- Auto speakers to give audio feedback to the driver, and to attend requirement R02.1,
- And the vehicle systems, to attend to requirements R04.3 and R04.3.2. They will be controlled by the developed system (output) while also giving feedback to it (input).

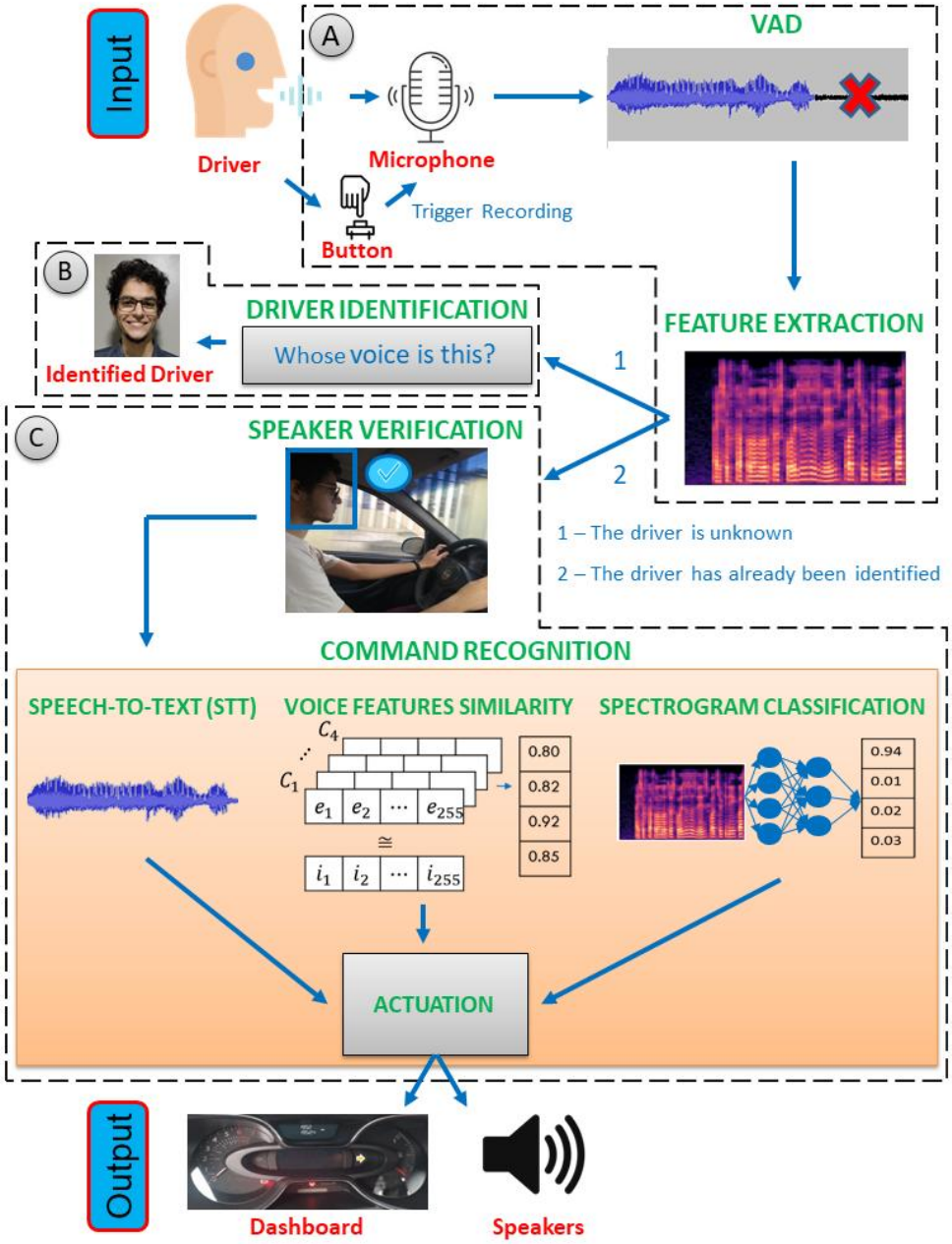
Figure 11 – Input-Output flow of the voice assistant system.



Source: Own authorship (2022).

In terms of functions, the system needs to detect speech in the recorded audio, identify the driver through its voice, verify if who is speaking is the driver, recognize voice commands, and act on the vehicle functions, as shown in Figure 12. The system was divided into three main subsystems: (A) Voice Activity Detection (VAD), (B) Driver Identification, and (C) Commands Recognition.

Figure 12 – Voice assistant information flow.



Source: Own authorship (2022).

3.1.2.1 System tests planning

The system test planning was elaborated based on the system needs. It should be followed in the system tests phase to verify if the system expectations were met. This planning was created as a checklist, presented in Frame 2, and each item should be tested and checked to ensure the system's usability.

Frame 2 – Checklist of planned system tests.

Test Case	Req. ID	Expected Result	Status
ST01	R02	The system indicates when it is activated	☐
ST02	R01.3	The system records audio only when required	☐
ST03	R03.1	The system verify if someone spoke	☐
ST04	R03.1.1	The system reports when no speech was detected	☐
ST05	R03.2	The system identify the driver through its voice	☐
ST06	R03.2.1	The system reports when the driver was not identified	☐
ST07	R03.2.2	The system reports when it identifies the driver	☐
ST08	R03.3	The system verify if the driver spoke	☐
ST09	R03.3.1	The system reports when the voice fails the verification	☐
ST10	R04.1	The system recognize the voice commands	☐
ST11	R04.1.1	The system reports if a voice command was not recognized	☐
ST12	R04.2	The system does not actuate if the command was not said by the driver	☐
ST13	R04.3	The system actuate if the command was recognized	☐
ST14	R04.3.1	The system reports when a command was received	☐

Source: Own authorship (2022).

These tests were designed with the user in mind, paying attention to system performance and user experience beyond the system's functioning.

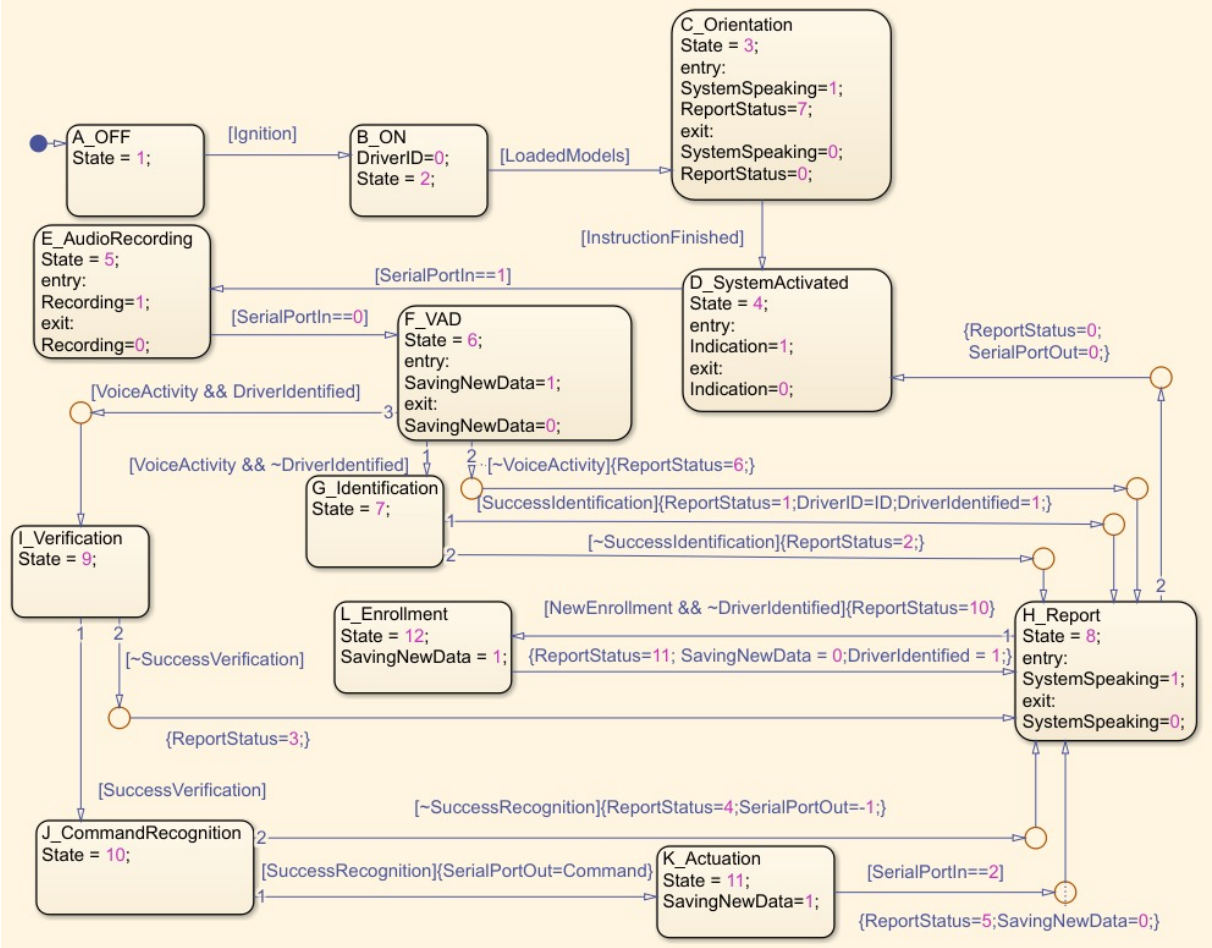
The test case column presented in Frame 2 refers to the identification of each test. The Requirements ID (Req. ID) column refers to the system requirement that is being tested. The status column represents the test report, which can be pass (✔) or fail (✘).

All tests were verified by simulating the test cases with Simulink Test Manager, to ensure that the system requirements could be tested after the development. The tests were described as an assessment in Simulink Test Manager. For example, the test case ST02, whose expected result is "The system records audio only when required", was described as "At any point of time, whenever ButtonPressed = 1 is true then, with no delay, Recording == 0 must be true".

We developed a state machine using Simulink Stateflow to simulate the tests and check if they attend to the expected results. The state machine is shown in Figure 13, whose states are represented by capital letters from A to L, followed by their name. Each state is connected to at

least one state, with a condition for the transition happening. For example, to pass from state A (system off) to state B (system on), the vehicle should be started through the ignition system.

Figure 13 – Voice Assistant Stateflow.

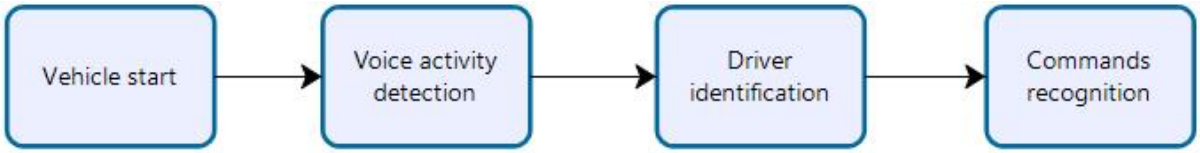


Source: Own authorship (2022).

3.1.3 Architecture design

Next, the system architecture was defined, as well as the specifications of its components and interfaces. Figure 14 shows the system's main subsystems and the vehicle start, even not being a subsystem of the developed system. It is only to indicate that the VA system starts with the vehicle start-up.

Figure 14 – Voice assistant system flowchart.

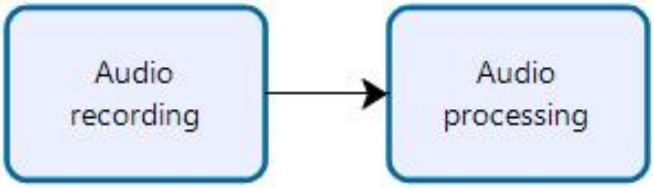


Source: Own authorship (2022).

Right after the vehicle start, the VA system should initialize and report to the driver, asking to identify who is driving by name, instructing the driver how it should be used, and asking to press the button and say his or her name.

In the voice activity detection phase, which process is presented in Figure 15, the user should hold the button as long as it takes to say his or her name. A microphone should record the audio, which will serve as input to the voice activity detection module. The recorded audio should be processed, and if it contains voice, the audio is passed to the next subsystem. Otherwise, the system should inform that no voice was detected in the recorded audio, asking the driver to speak again.

Figure 15 – Voice activity detection process details.

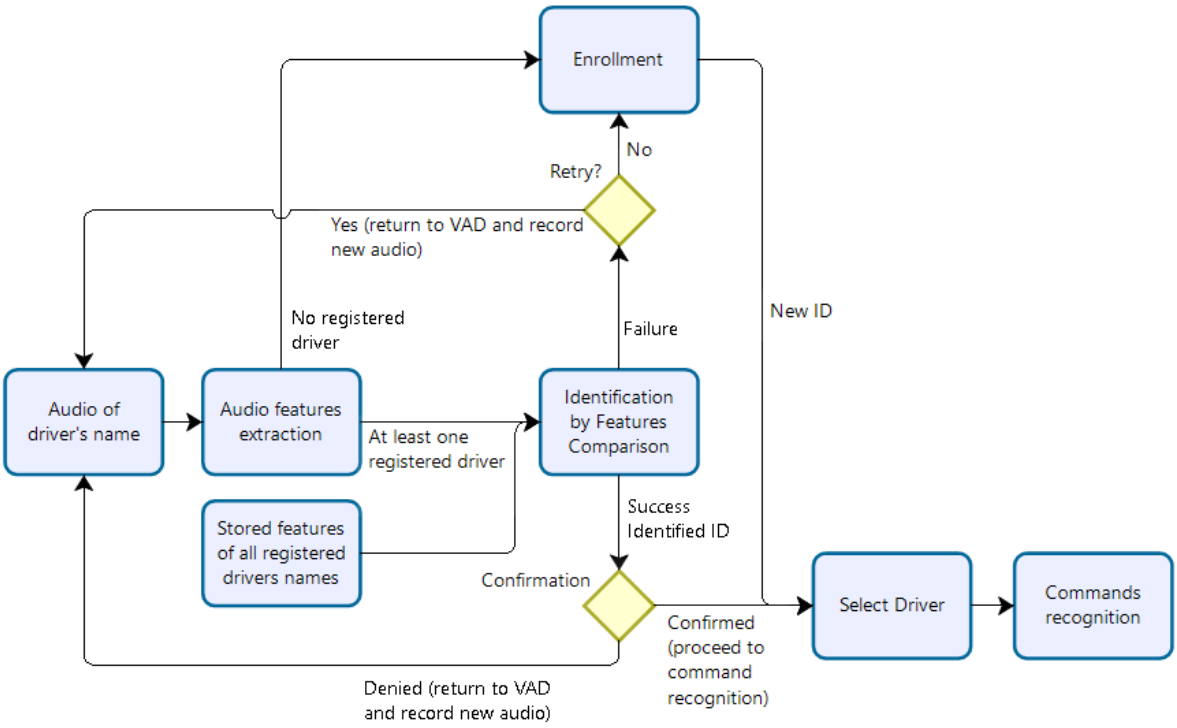


Source: Own authorship (2022).

The driver identification subsystem should identify the driver through its voice in the input audio. The process flow is detailed in Figure 16. The driver identification process should check if there is at least one registered driver. If not, the enrollment process should start, receiving the same audio as input. If there is at least one registered driver, the input audio features should be extracted and compared to the enrolled voices to verify if the actual speaker is registered and recognize its identification (ID) number. The yellow diamonds indicate decisions from the system that need confirmation from the user.

The system should ask the driver to confirm the identified number by holding the same button used for recording. The system must start the command recognition subsystem if the driver confirms the ID. Otherwise, the system should retry the identification. If the identification fails, the system should ask if the driver wants to retry the identification or start enrolling a new driver.

Figure 16 – Driver identification process details.

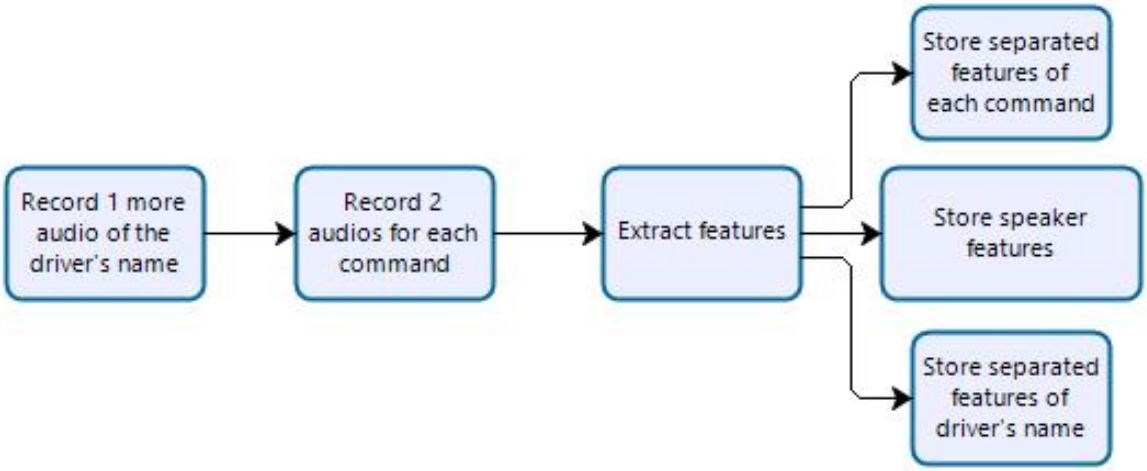


Source: Own authorship (2022).

The output of the driver identification subsystem should be a signal to activate the command recognition subsystem, which should happen if the driver was identified as one of the registered IDs or after assigning a new ID to the driver in the enrollment process.

The enrollment process is detailed in Figure 17. It should be called by the driver identification subsystem when the driver is not identified.

Figure 17 – Enrollment process details.



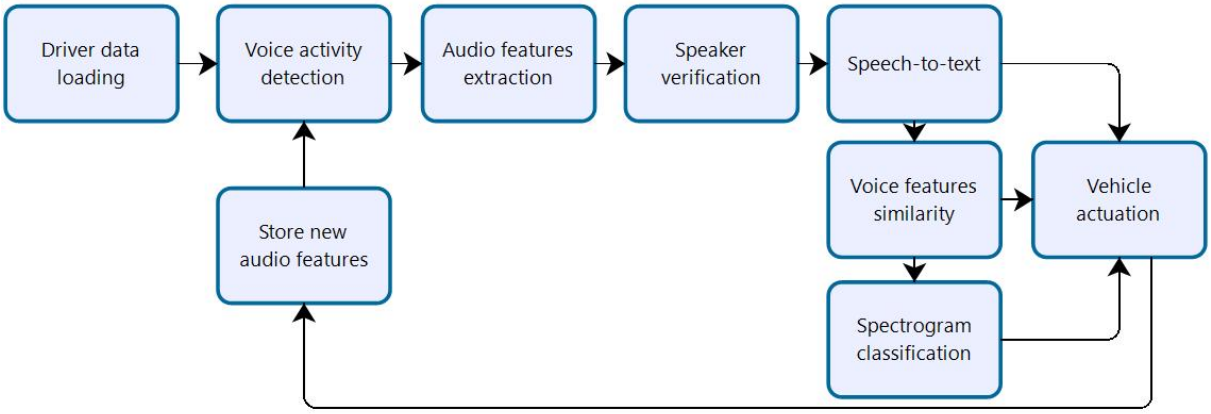
Source: Own authorship (2022).

The system should orient the driver (through the auto speakers) to record another audio saying its name. Additionally, the system should inform which will be the driver's ID number. After that, the system should instruct the driver to say all the voice commands two times each. The recordings of these voice commands and the driver's name should then pass to a feature extraction module. These features should summarize the characteristics of the recorded audio. The extracted features of each command and of the driver's name audios should be stored in a file. After that, a file should summarize the driver's voice. This module should also extract the spectrogram of each audio and save them to image files.

A total of 6 files containing the audio features should be generated and stored, one for each voice command (totaling 4), one for the driver's name, and one for the general features of the driver's voice. Additionally, eight spectrogram images should be obtained, one for each voice command audio. These files will serve as inputs for the driver identification and commands recognition subsystems.

The commands recognition process flow is presented in Figure 18. It should start after the driver identification success or the enrollment of a new driver, and it runs in a loop until the driver turns the vehicle off. Right at the start of this subsystem, the VA should inform the driver that it is ready to receive voice commands reminding the user how to use the system.

Figure 18 – Commands recognition process details.



Source: Own authorship (2022).

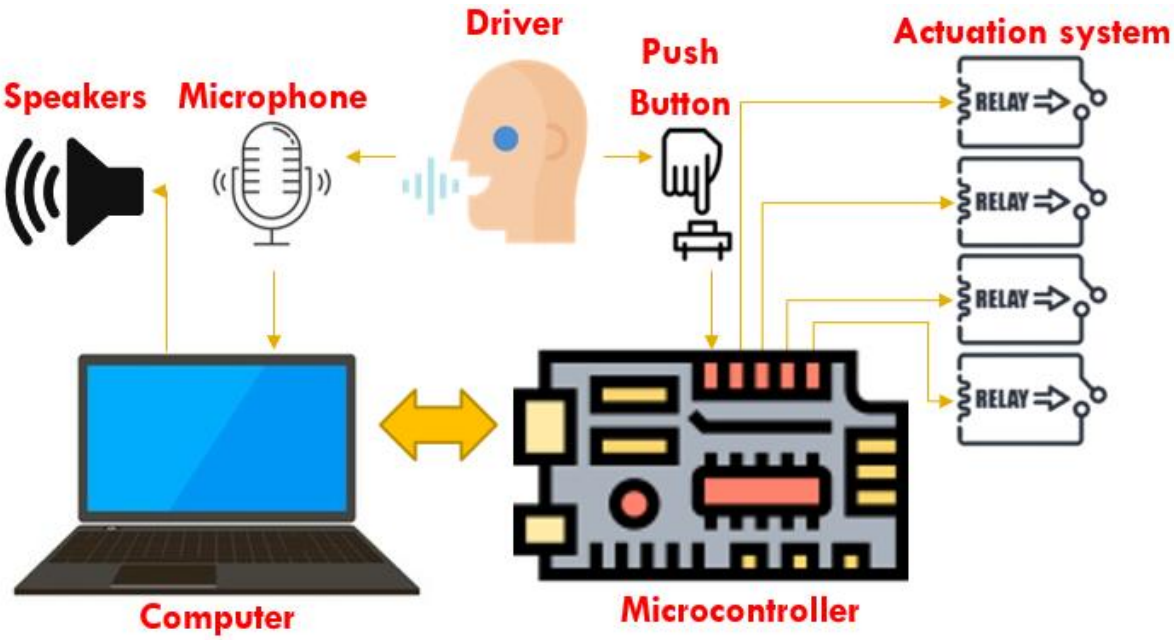
The driver should record one new audio, saying a voice command to start the subsystem. Then, the voice activity detection module should try to detect speech in the recorded audio. After that, the recorded audio passes through the speaker verification subsystem to verify if the voice is from the driver identified at the start of the VA system. Then, three different techniques should try to recognize the voice command to ensure that the recognition is reliable. The recognition

techniques are Speech-to-Text, Voice Features Similarity, and Spectrogram Classification. The system should inform the user and act on the vehicle if it recognizes the command. Finally, the audio features should be stored, and the system should be ready to receive new commands.

3.1.3.1 Hardware architecture

The hardware needed for the system to work differs depending on the system's version. For the first one, only a computer with a microphone and auto speakers (or a headset) is required. For the second version, the hardware needed consists of a button, a microphone, auto speakers, a microcontroller to simulate the vehicle signals, and a dedicated computer to run the VA software. The hardware architecture of the second version of the prototype is represented in Figure 19.

Figure 19 – System prototype (V2) hardware architecture.



Source: Own authorship (2022).

3.1.3.2 Integration tests planning

As well as the system tests, the integration tests were planned to ensure the architecture requirements. The tests are presented in Frame 3 and should be performed after the components tests phase to check the subsystems and their integration.

Frame 3 – Checklist of planned integration tests.

Test Case	Req. ID	Expected Result	Status
IT01	R01	The microphone picks up audio for the computer	☐
IT02	R02.2	The computer outputs audio through the speakers	☐
IT03	R01.2	The push button receives 3.5V or more from the microcontroller	☐
IT04	R01.2	The push button outputs 1V or less to the microcontroller when not pressed	☐
IT05	R01.2	The push button outputs 3.5V or more to the microcontroller when pressed	☐
IT06	R04.3.2	The computer reads a 16 bits signal by serial port from the microcontroller actuation system	☐
IT07	R04.3	The actuation system relays receives 12V or more from the car battery	☐
IT08	R04.2	The actuation system relays output 1V or less when not activated	☐
IT09	R04.3	The actuation system relays output 12V or more when activated	☐
IT10	R04.3	The microcontroller receives a 16 bits signal by serial port from the computer actuation system	☐

Source: Own authorship (2022).

In terms of hardware architecture, the first version of the prototype does not need integration tests since the only hardware is a computer, in the case of using an internal microphone and embedded speakers, whose architecture has already been validated. In the other case (external microphone and speakers), the connections to the computer should be tested according to Frame 3.

3.1.4 Components design

The hardware and software components of the system were established and detailed in the final design phase before the implementation. That way, their specifications should be followed in the implementation phase.

3.1.4.1 Hardware components

The hardware needed for the first version of the system consists of a computer. We opted to use a notebook (whose specifications are presented in Frame 4) since it owns an embedded microphone and built-in speakers, and its architecture has already been tested and validated, as well as its components. The button necessary for the system can be any key from the notebook keyboard.

Frame 4 – Specifications of the personal computer used for the system prototype.

Component	Specification
Microprocessor	Intel® Core™ i5-8250U (1.6GHz, up to 3.4GHz) 6MB Cache
Memory, standard	8GB, DDR4 2400MHz (2 x 4 GB)
Video Graphics	Intel UHD Graphics 620
OS	Windows 10
Storage	1 TB HDD and 256 GB SSD (M2)

Source: Own authorship (2022).

For the second version of the system prototype we used the same computer, connected to a microcontroller, which had a button as input, and Light-Emitting Diode (LED) and relays as output. For this version of the prototype, we decided to use an external microphone.

The button needed for the audio recordings and for user interaction with the system, should be a push button in the normally open configuration, to correctly communicate with the microcontroller program. It should have a raised actuator. In addition, it must be positioned in a location that is easily accessible to the driver.

The microphone should have cardioid polar pattern, as it has a narrow coverage angle and is more sensitive to audio coming from the front of the microphone.

When it comes to the auto speakers, no specific configuration is necessary, the vehicle's own speakers can be used.

3.1.4.2 Software components

At the start of the system, the introduction module should introduce the assistant to the user and instruct how to use it through the driver report module. This module should be called right after the system finishes loading all dependencies, and it should invoke the voice activity detection subsystem, which begins with the audio recording module.

The driver report module should inform the driver of any failure during the VA subsystems, like not recognizing the voice command, not detecting speech in the recorded audio, or not identifying the driver by voice. Additionally, this module should inform the driver when it has successfully recognized a voice command or identified the driver. This module should give feedback to the driver on what is happening to the system. The input should be a list of messages from the system, and the output should be synthesized audio.

3.1.4.2.1 Voice activity detection subsystem modules

The audio recording module should wait for the user to press the button to start capturing the audio. The audio capture should last while the user is holding the button and have a maximum duration of 4 seconds. The audio signal should be saved in a *.wav* file. The audio input of this module should come directly from a microphone or pass through a DSP device for noise filtering.

After the audio recording module, the recorded audio serves as input to the audio processing module, which should try to detect speech in the audio and pass the audio to the next

module. If the module does not detect speech, it should call the driver report module.

3.1.4.2.2 Driver identification subsystem modules

The driver identification subsystem starts with the audio features extraction module, which should receive one audio, in the *.wav* file format, as input. This module should extract an embedding from the input audio, summarizing the audio features. Then, save it in an array in a *.npy* file. This module should also generate the audio spectrogram and store it on a *.png* file.

The features comparison module should check the folder of registered drivers, looking for all *.npy* files containing embeddings of the drivers' names. This module should receive the embeddings of the driver's name and compare its similarity to all the found embeddings. If the highest similarity is higher than a threshold (to be defined), a confirmation from the driver is requested, and if approved, the commands recognition subsystem is started. Otherwise, the module should ask if the driver wants the system to retry the identification with new audio or if the enrollment process should begin.

The enrollment process should use the module of audio recording and the audio features extraction module already defined, as well as the features storage module, which in turn, should verify the next driver ID available, create the folders relative to this ID and finally store each *.npy* and *.png* file, received from the audio features extraction module, on their respective folders.

The select driver module should receive a driver ID as input and return the path to the folder, which contains all data from the driver produced and stored in the enrollment phase.

3.1.4.2.3 Commands recognition subsystem modules

The driver data loading module should load all the driver embeddings stored in *.npy* files. This module should receive the driver folder path and driver ID as inputs and output an array with the embeddings.

The speaker verification module should receive the recorded audio embedding and the driver embeddings array and compare their similarity to check if the voice, which claims to be from the driver, really is. For this, the similarity of the embeddings must be greater than a threshold. The module's output should be a logical value relative to the similarity comparison to the threshold.

The speech-to-text module should receive the recorded audio *.wav* file and a STT model.

It should return a text with the most likely command based on the audio transcription ASR evaluation metrics and two logical values, indicating if the transcript is identical to a command or not and if at least one word of the transcript matches with the estimated command.

The voice features similarity module should receive the audio embedding, the driver embeddings for each voice command, and the written estimated command from the speech-to-text module. It should compare the audio embedding with the driver commands embeddings to calculate the similarity. The module should output the command with the highest similarity score, the highest similarity score, and the similarity score of the estimated command from the STT module.

The recorded audio spectrogram file and a classifier model are the inputs of the spectrogram classification module, which, in turn, should classify the spectrogram into one of the commands and return the predicted class (command with the highest score) and the relative score.

The vehicle actuation module should receive the results of the three command recognition modules. The system should be aware of the vehicle systems' current state, so it should verify them before the decision-making. Based on the command recognition modules (speech-to-text, voice features similarity, and spectrogram classification), this module should decide which vehicle system to activate (or deactivate) and then inform the user through the driver report module. The outputs of this module should be an electrical signal to the vehicle system and the resultant command text to the features storage module.

The new features storage module should receive the written command, the embeddings, and the spectrogram from the last recorded audio. This module should save the embeddings in the corresponding *.npy* file and store the spectrogram image file in the appropriate folder.

3.1.4.3 Components tests planning

Each component of the system should be tested to ensure that all of them work well independently, avoiding error propagation and reducing the time spent during the validation of the entire system. That way, Frame 5 shows the tests that need to be made, after the implementation step, to check if all the modules are working correctly.

Frame 5 – Checklist of planned components tests.

Test Case	Req. ID	Expected Result	Status
CT01	R02.3	The introduction module explains how the button should be used to interact with the system	☐
CT02	R01.3	The audio recording module does not record when the button is not being pressed	☐
CT03	R01.3	The audio captured by the microphone is recorded to a *.wav* file	☐
CT04	R03.1	The audio processing module segments the audio into speech and non-speech sections	☐
CT05	R02.2	The driver report module speaks the input text	☐
CT06	R03	The audio features extraction module saves a *.npy* file with the input audio embeddings	☐
CT07	R04.1	The audio features extraction module saves a *.png* file with the input audio spectrogram	☐
CT08	R03.2	The features comparison module calculates the similarity of the audio features with a list of *.npy* files	☐
CT09	R03.4	The features storage module creates the needed folders	☐
CT10	R03.4	The features storage module saves each file to the corresponding folder	☐
CT11	R03.4	The select driver module finds the driver folder relative to the given ID	☐
CT12	R03.4	The driver data loading module loads all the *.npy* files from the driver folder to an array	☐
CT13	R03.3	The speaker verification module recognizes when the voice is not from the loaded driver	☐
CT14	R04.1	The STT module estimates the transcript of the audio	☐
CT15	R04.1	The STT module calculates the ASR evaluation metrics	☐
CT16	R04.1	The STT module outputs a single command	☐
CT17	R04.1	The voice features similarity results in an array with 4 similarity scores, one for each command	☐
CT18	R04.1	The voice features similarity module outputs a single command	☐
CT19	R04.1	The spectrogram classification module outputs a single command	☐
CT20	R04.3.2	The vehicle actuation module reads the simulated signals from the vehicle	☐
CT21	R04.3	The vehicle actuation module activate/deactivate the chosen vehicle system	☐
CT22	R03.4.1	The new features storage module appends the embeddings array to the chosen *.npy* file	☐
CT23	R03.4.1	The new features storage module saves the spectrogram image file	☐

Source: Own authorship (2022).

3.2 Voice assistant implementation

The implementation phase began by creating a dataset of voice commands, training all the necessary ML models, developing the modules, and finally, integrating these modules and the subsystems. The system's flowchart is shown in Figure 14, containing the subsystems that will be further explained.

The VA system was divided into three subsystems:

- Voice activity detection,

- Driver identification,
- And commands recognition.

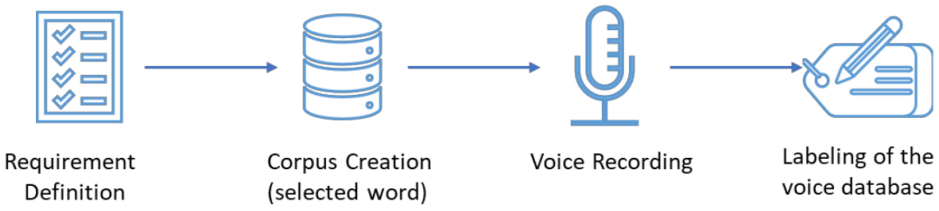
3.2.1 Dataset creation

An audio dataset should have audio files and a file containing the paths to the audio files and the audio transcripts. Additionally, the gender and age of the speakers may be included. Furthermore, if there are many languages in the dataset, the language used should be described, as well (FENDJI *et al.*, 2022).

For this project, we decided to focus on the recognition of specific voice commands in BP. As a result, the VA does not have to understand everything the driver and passengers say. Since none of the free BP voice datasets available on the internet met the project's needs, a new dataset was created with voice commands to control functions of the dashboard of a vehicle.

The creation of the voice commands dataset was based on the steps presented by Fendji *et al.* (2022), shown in Figure 20.

Figure 20 – Steps of creation of a limited vocabulary audio dataset.



Source: Fendji *et al.* (2022).

3.2.1.1 Requirement definition

For the construction of the dataset, different people should record audios of each command, preferably men and women, with different tones of voice. Each person should record three audios of each command.

The audios should have a fixed duration of 4 seconds and can be recorded using any computer or smartphone equipped with a microphone and connected to the internet. The sample rate defined for the recordings was 16 kHz.

As the project was meant to be applied to Brazilian vehicles, the language chosen for the dataset was Brazilian Portuguese.

3.2.1.2 *Corpus creation*

A list of commands was created in order to be used as a small dictionary for developing a fast, accurate and offline ASR system to recognize voice commands from the vehicle driver.

Frame 6 presents the translation and the description of what the commands should trigger in the vehicle since they are in Brazilian Portuguese.

Frame 6 – Voice commands translation and vehicle function.

Brazilian Portuguese	English	Vehicle Function
Seta para direita	Right turn signal	Turns on/off the right turn signal
Seta para esquerda	Left turn signal	Turns on/off the left turn signal
Luz baixa	Headlights	Turns on/off the headlights
Pisca alerta	Hazard warning	Turns on/off the hazard warning lights

Source: Own authorship (2022).

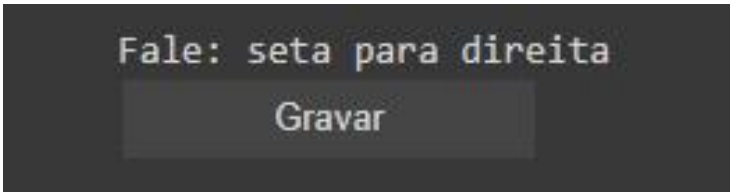
3.2.1.3 *Voice recording and dataset labeling*

We developed an application called Audio Dataset Creator using Google Colaboratory (Colab) environment and Python programming language for the voice recording step.

The use of Google Colab was to allow anyone to participate in the data acquisition step. The only requirement was a device with an internet connection and a microphone. This was planned to improve people's acceptance of the voice recording process.

The application guides the user through recording 3 audios of each command, totaling 12 audios, each with 4 seconds. The process of recording is straightforward and should take approximately 1 minute. The user needs to run all the cells of the runtime and then follow the instructions of which command should be spoken after pressing the record button, which is shown in Figure 21. The user can include age and gender, but it is not mandatory. Figure 21 also shows the command that should be spoken: *seta para direita* (Right turn signal), above the button *Gravar* (Record).

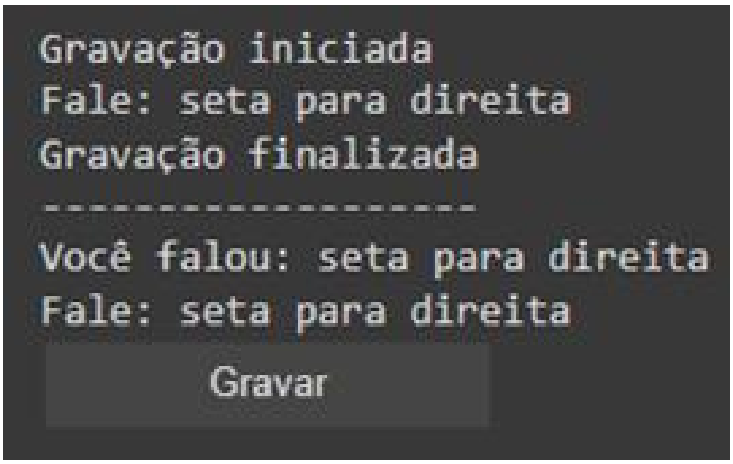
Figure 21 – Record button of the Audio Dataset Creator.



Source: Own authorship (2022).

After each recording, the application verifies if the audio should be included in the dataset through speech recognition using the Python library SpeechRecognition, which supports engines and APIs, online and offline. The verification process is shown in Figure 22, and consists in comparing the text of the command shown on the user screen and the transcript obtained from the speech recognition of the recorded audio.

Figure 22 – Audio Dataset Creator automatic verification.



Source: Own authorship (2022).

If the texts are equal, the application saves the audio with the corresponding label and requests the next recording from the user. If they are different, the application asks the user to record again. After all recordings, the application compresses their files and downloads a single *.zip* file.

The speech recognition engine chosen from the library was Google Speech Recognition since it supports Brazilian Portuguese with good STT results in previous tests of the recorder application.

3.2.2 Speech-to-Text

One of the command recognition subsystems consists in STT, which is used to convert audio to text through an acoustic model, which needs to be trained before it can be used for inference on new data. It can also use a language model to improve the quality of the transcriptions.

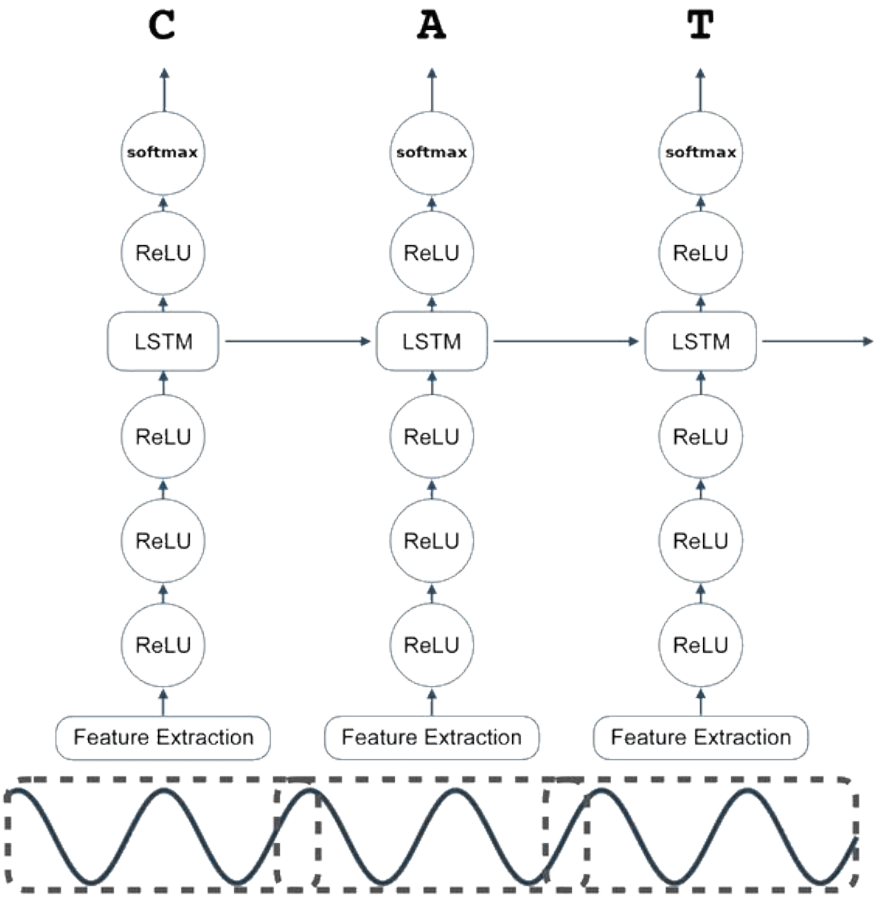
The following sections will explain how these models were trained, the AM in Subsection 3.2.3 and the LM in Subsection 3.2.4.

3.2.3 Acoustic model

We trained a Portuguese AM using the open source engine DeepSpeech provided by Mozilla, which is based on the paper of Hannun *et al.* (2014) and uses Google's TensorFlow. The training data for an AM is a corpus of speech and transcripts.

According to Morais and Davis (2020), they trained an RNN to generate English text transcriptions from speech spectrograms. An example of inference with the resulting RNN model, created by Morais and Davis (2020), is presented in Figure 23.

Figure 23 – Example of inference with the complete Recurrent Neural Network (RNN) model of DeepSpeech.



Source: Morais and Davis (2020).

3.2.3.1 Dataset preparation

For the preparation of the dataset, the audios and their corresponding transcripts were split into training, validation, and test. Both validation and test splits correspond to 15% of the

dataset, and the training files correspond to the remaining 70%. The rest of the data containing age and gender was not used for the AM training.

Besides that, the transcripts passed through a preprocessing step consisting of removing special characters, changing all texts to lowercase, and replacing characters with accentuation.

Table 2 shows the number of audios of each command in each split set (training, validation, and test). Table 3 shows the number of audio files of each command present in each dataset subdivision in the mixed dataset, which was complemented with 16 artificial voices from Microsoft Azure TTS service, totaling 192 additional voice commands, with three audios of each command, varying the speaking speed in 85%, 100%, and 115%. This dataset was named VVCMD because it mixed the original dataset with synthesized voices. The dataset recorded as described in Subsection 3.2.1 was named Vehicle Voice Commands Dataset (VVCD).

Table 2 – Number of audio files of each command present in each subdivision of the recorded dataset.

	Right turn signal	Left turn signal	Headlights	Hazard warning
Training	36	38	37	31
Validation	8	7	7	9
Test	7	6	7	11
Total	51	51	51	51

Source: Own authorship (2022).

Table 3 – Number of audio files of each command present in each subdivision of the mixed dataset.

	Right turn signal	Left turn signal	Headlights	Hazard warning
Training	73	71	66	67
Validation	13	14	15	17
Test	13	14	18	15
Total	99	99	99	99

Source: Own authorship (2022).

3.2.3.2 Training

The training process was done using Transfer Learning in Google Colab due to the available Graphics Processing Unit (GPU) that can accelerate the training of machine learning models. The GPU used for training was the NVIDIA Tesla T4.

Frame 7 presents the parameters used for the training.

Frame 7 – Acoustic model training parameters.

Parameter	Value
Epochs	100
Early stop	True
Early stop epochs	25
Learning rate	0.0001

Source: Own authorship (2022).

The network had 2048 hidden layers and used the Adam optimizer (KINGMA; BA, 2014) with the default parameters. Automatic mixed precision was also used since it can speed up the training process. Besides that, the Allow Growth flag was enabled to ensure that only the required amount of GPU memory would be allocated and prevent the full allocation of available GPU memory.

Four different training configurations were tested, varying two parameters on the training settings: the used dropout and the use, or not, of data augmentation. These configurations are shown in Table 4.

Table 4 – Different training configurations for the acoustic model.

AM	Dropout rate	Data augmentation
1	0.05	No
2	0.05	Yes
3	0.2	No
4	0.2	Yes

Source: Own authorship (2022).

These four training sessions were made using the VVCD dataset to determine the best settings before training in a larger dataset, which is the case of VVCMD.

We planned the training to last 100 epochs and to stop earlier if there were no significant improvements in the validation loss in the past 25 epochs. The value used as the minimum change in loss to qualify as an improvement was 0.05, which is the default value in DeepSpeech settings.

In addition to loss, two ASR metrics were computed during the model training: WER and Character Error Rate (CER). According to Morris *et al.* (2004), WER is calculated using Equation 6.

$$WER = \frac{S + D + I}{N_1 = H + S + D}, \tag{6}$$

where S is the number of substitutions, D is the deletions, I is the insertions, H is the number of hits (correct words), and N_1 is the total words in the source sentence. The HSDI counts are relative to the comparison of the obtained transcript (output) and the source sentence (input).

The CER is calculated similarly, but at the character level, counting the HSDI of characters in the sentence.

3.2.3.2.1 Data augmentation

We used nine distinct data augmentation techniques on the voice command audios to supplement the data. According to DeepSpeech training documentation (MORAIS *et al.*, 2020), each data augmentation technique available targets a unique domain, being applied in the order:

1. Sample domain,
2. Signal domain,
3. Spectrogram domain,
4. Features domain.

All augmentations were applied to the audios with a probability of 10%. The augmentations of the sample domain were reverb and volume. The signal domain augmentations were the time mask and the add. In the spectrogram domain, the augmentations employed were: pitch, tempo, frequency mask, and dropout. Finally, multiply augmentation was used in the features domain.

The reverb augmentation was applied with delay varying from 20 ms to 80 ms and decay starting with a random number between 11 dB and 9 dB at the beginning of training and going to a random number between 3 dB and 1 dB at the end of training. This augmentation applies a Schroeder reverberator with no all-pass filters.

The volume augmentation started the training with a target volume of -10 dBFS (Decibels relative to full scale) and finished at -40 dBFS. It measures and levels the samples.

At the start of training, the time mask applied 1 to 5 intervals of silence in the signal with durations ranging from 10 ms to 90 ms, increasing to 8 to 12 intervals with durations ranging from 60 ms to 140 ms at the end of the training.

The add augmentation used a standard deviation of the normal distribution with a mean of 0, ranging from -0.5 to 0.5. Since it was applied in the signal domain, it will add random values inside the standard deviation to the sample waveform.

The pitch augmentation scales the frequency axis of the spectrogram. A pitch factor ranging from 0.8 to 1.2 was used.

The tempo augmentation scales the time axis of the spectrogram. A speed factor ranging from 0.5 to 1.5 was used.

The frequency mask applied one interval of silence in 1 frequency band at the beginning of training, increasing to 3 intervals in 5 frequency bands at the end of training.

The dropout augmentation was applied to the spectrogram with a 5% dropout rate, which means it will zero 5% of the data, chosen randomly, from the spectrogram.

The multiply augmentation used a standard deviation of the normal distribution with a mean of 1, ranging from -0.5 to 0.5. Since it was applied in the feature domain, it will multiply random values inside the standard deviation to the sample's mel-spectrogram features.

3.2.4 Language model

To improve the accuracy of transcription, an LM should be used to calculate the probability of a sequence of words happening. It is also called scorer, and the default used by DeepSpeech was trained on the LibriSpeech dataset (MORAIS, 2020) in which the data is from audiobooks in English from the LibriVox project (PANAYOTOV *et al.*, 2015).

We trained a new scorer since this project aims to develop a voice assistant in Brazilian Portuguese. The training data for an LM is a corpus of text.

3.2.4.1 Dataset preparation

We used three different corpora as training data:

The first one was the MLS dataset (PRATAP *et al.*, 2020), which consists of a large multilingual corpus created for speech research. It is derived from audiobooks from LibriVox, just like the LibriSpeech dataset used for training the DeepSpeech LM. The Portuguese dataset is from 460 books with 1.24 million sentences and 13.84 million words.

The second was the Europarl corpus (KOEHN, 2005) is extracted from the proceedings of the European Parliament and includes versions in 21 European languages: Romanic (including Portuguese), germanic, Slavik, Finni-Ugric, Baltic, and Greek. The Portuguese in this dataset is from Portugal, which has some differences from Brazilian Portuguese.

The final corpus was the WikiText PT-BR dataset (QUINTANILHA *et al.*, 2020), built for a Ph.D. work in progress. It is a collection of over 8 million sentences extracted from Wikipedia articles in Brazilian Portuguese. The dataset was pre-processed to remove all

punctuation, and the numbers were converted into their written form.

The number of occurrences of each word present in our dataset of commands is shown in Table 5. Each word probability is shown in parentheses and was calculated by Equation 3. The highest probability of each word is represented in bold.

Table 5 – Word count in each used corpus.

Word	MLS	Europarl	WikiText PT-BR
Alerta	21 (1.52E-6)	**951 (1.90E-5)**	2187 (1.12E-5)
Baixa	802 (5.80E-5)	1145 (2.29E-5)	**22251 (1.14E-4)**
Direita	**1256 (9.08E-5)**	863 (1.73E-5)	12291 (6.31E-5)
Esquerda	912 (6.59E-5)	1003 (2.01E-5)	**14740 (7.57E-5)**
Luz	**5637 (4.07E-4)**	4095 (8.20E-5)	26075 (1.34E-4)
Para	104982 (7.59E-3)	**531769 (1.06E-2)**	1682283 (8.64E-3)
Pisca	**10 (7.23E-7)**	0 (0)	110 (5.65E-7)
Seta	**61 (4.41E-6)**	1 (2.00E-8)	449 (2.31E-6)

Source: Own authorship (2022).

3.2.4.2 Training

We trained the language model using KenLM (HEAFIELD, 2011). A 5-gram model was initially used, which considers the four words that precede the word w_i to calculate its probability. We selected a 5-gram according to the DeepSpeech language model training guide (MORAIS, 2020). We used Google Colab for training with the default settings presented in (MORAIS, 2020), except for the top-k, which we set to 50000 because the number of unique words in the first two datasets was less than 500000, which was the default value for k. The default settings included pruning singleton n-grams of order 3 and higher.

We added all written commands to every corpus 100000 times to increase the probability of each word. The probabilities should be further analyzed to decide which is the most suitable dataset to be used as a training dataset. We trained a 5-gram, a 4-gram, a trigram, and a bigram with the selected dataset, to assess the effects of the n-gram model's order.

3.2.5 Spectrogram classifier model

We trained a spectrogram classifier model using transfer learning on 15 different image classification models using Tensorflow Keras API.

3.2.5.1 *Dataset preparation*

The dataset used for training the spectrogram classifier was the original one (VVCD), containing only the voices of the 17 volunteers. All audios were converted to images by extracting the mel-spectrograms with the Librosa python package. All images are similar to Figure 9. Table 6 shows the distribution of images in training, validation, and test sets.

Table 6 – Number of image files of each command present in each dataset subdivision in the dataset.

	Right turn signal	Left turn signal	Headlights	Hazard warning
Training	35	35	35	35
Validation	8	8	8	8
Test	8	8	8	8
Total	51	51	51	51

Source: Own authorship (2022).

3.2.5.2 *Training*

We executed the training using Google Colab and planned it to last 150 epochs, with early stopping based on the validation accuracy, with a patience value of 15, which means that if the validation accuracy does not improve in 15 epochs, the training finishes early. The Adam optimizer was used with the categorical cross-entropy loss. Validation loss was also used as the early stopping criterion.

We used Imagenet weights as the starting point for the image classification training and trained using the transfer learning technique on 15 Keras models.

The Keras API has 33 available models for classification and feature extraction applications. Table 7 presents the 15 models selected for the spectrogram classifier training. The time column is relative to the time in milliseconds needed by the GPU per inference step. While MobileNetV2 shines for being the smallest, MobileNet stands out as the fastest.

Table 7 – Keras available models.

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms)
Xception	88	79.0%	94.5%	22.9M	81	8.1
VGG16	528	71.3%	90.1%	138.4M	16	4.2
VGG19	549	71.3%	90.0%	**143.7M**	19	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	**449**	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	**3.4**
MobileNetV2	**14**	71.3%	90.1%	3.5M	105	3.8
DenseNet201	80	77.3%	93.6%	20.2M	402	6.7
EfficientNetB7	256	**84.3%**	**97.0%**	66.7M	438	61.6

Source: Adapted from Keras Team (2022).

3.2.6 Software development

We used the Python language to develop the VA software, as it is widely used for ML and Artificial Intelligence (AI) applications, due to its rich set of libraries and frameworks like TensorFlow and Keras, which we used to train the necessary models.

The driver report module was created using pyttsx3, a python TTS library that works offline and supports multiple TTS engines. We used the Brazilian Portuguese female voice of Microsoft Windows.

The introduction module was programmed to call the driver report module to explain how the assistant should be used.

3.2.6.1 Voice activity detection subsystem

For the VAD, the audio recording module was developed using the PyAudio library to record audio with the computer microphone and the wave module to write the audio to a *.wav* file. The audio recording starts when the user presses a button on the keyboard, which was achieved using the keyboard python library.

We programmed the audio processing module using webrtcvad, a python interface to the Google WebRTC VAD. The aggressiveness was set to 3 so that it filters more non-speech segments of the audio. At least 20% of the audio must be of speech segments. Otherwise, the

audio is considered non-speech audio and is discarded. This module is capable of saving new audio containing just the speech segments.

3.2.6.2 *Driver identification subsystem*

The audio features extraction module was developed using Librosa (MCFEE *et al.*, 2020) for the mel-spectrogram extraction and Resemblyzer for the audio features embedding. Resemblyzer is a python package that uses a voice encoder, which is a deep learning model, to analyze and compare voices. The voice encoder from Resemblyzer is described by Jemine (2019). The module creates an L2-normed array with 256 values summarizing voice features and stores it in a *.npy* file so it can be used by other modules. It also saves the spectrogram to a *.png* file with the same purpose.

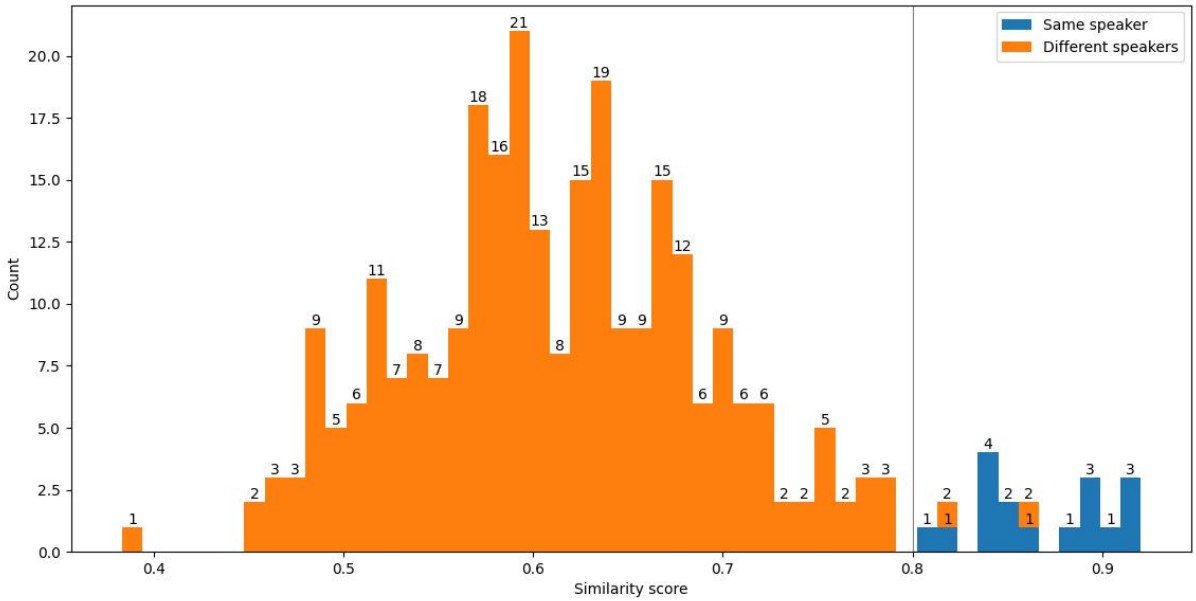
Since all embeddings pass through L2 normalization, their magnitudes are equal to 1, reducing the similarity calculation to the dot product of the two vectors, as shown in Equation 7.

$$cos(\theta) = \mathbf{A} \cdot \mathbf{B} = \sum_{i=1}^{n} A_i B_i \tag{7}$$

The stored array with the higher similarity to the newly recorded audio is selected as the possible driver by the driver identification subsystem, but only if the similarity is greater than the defined threshold of similarity.

The initial threshold was defined based on the histogram of similarity between speakers presented in Graph 1. According to the histogram, a good starting point is 0.8, as it separates well when it is the same speaker or different speakers. Similarities of audios from the same speaker are always greater than 0.8, and only two comparisons of audios from different speakers exceed this value. It may be updated during the components testing to improve the system's functioning.

Graph 1 – Histogram of similarity values between speakers.



Source: Own authorship (2022).

We generated this histogram using Resemblyzer with all audios of VVCD. We computed two speaker embeddings (A and B) per speaker, each with six audios of the twelve recorded. A speaker embedding is the average of its audio embeddings. Then, we compared these two speaker embeddings to generate the 17 similarities of the same speaker, and each speaker embedding A to all other speakers embeddings B to calculate the similarities of different speakers, totaling 272 similarities (${}^{17}P_2$).

3.2.6.2.1 Speaker enrollment

The speaker enrollment module guides the user to speak the voice commands. It creates a folder for each new driver to store its recorded audio, its embeddings, and spectrogram files. It records two audios of each voice command and two of the name of the driver. After each recording, the audio spectrogram is computed by the audio features extraction module.

After the recordings are over, all audio voice features are extracted and embedded in a single array, summarizing the driver's voice. This array is stored in a file named *driver.npy*. All pairs of voice commands also have their audio features embedded, each to a different *.npy* file. The audios of the driver name also generate a *.npy* file with its feature embeddings.

3.2.6.3 *Commands recognition subsystem*

The command recognition subsystem is considered the most important for ensuring the safety needed in the project since it can control some of the vehicle dashboard systems, which include but are not limited to headlights and turn signals.

We developed a three-step verification method to recognize the commands and correctly activate the vehicle functions. This system is composed by:

- A STT algorithm to translate the speech recorded into words and check if there is a command in the sentence,
- A Voice Features Similarity algorithm to check the similarity of the recorded audio with the driver commands recorded in the enrollment step,
- And finally, an image classification step, using CNN to classify a spectrogram generated through the recorded audio.

The recorded audio passes through the speaker verification module before entering the command recognition step to ensure the command was made by the driver. This module calculates the similarity of the recorded voice command embedding to the *driver.npy* file created in the driver enrollment. If the similarity is higher than a threshold, the audio is considered as spoken by the driver and goes to the command recognition first step, which is the STT.

3.2.6.3.1 Speech-to-Text

The STT module converts the input audio to text, which is considered the audio's transcript. We used the DeepSpeech engine with the trained acoustic and language models described in Subsection 3.2.3 and Subsection 3.2.4, respectively. The obtained transcript is then evaluated by Algorithm 1 to estimate which is the most probable command.

Algorithm 1 – Estimate Speech-to-Text command.

Input: $WER_{list}, CER_{list}, MER_{list}, WIP_{list}, WIL_{list}, D_{list}, C_{list}$
1: $WER_{list} = Normalize(WER_{list})$
2: $CER_{list} = Normalize(CER_{list})$
3: $maxWIP = Max(WIP_{list})$
4: $indexMaxWIP = Argmax(WIP_{list})$
5: $indexMinWER = Argmin(WER_{list})$
6: $indexMinCER = Argmin(CER_{list})$
7: **if** $maxWIP = 1$ **then**
8: $perfect =$ **true**
9: **else**
10: $perfect =$ **false**
11: **end if**
12: **if** $maxWIP > 0$ **then**
13: $trust =$ **true**
14: **if** $Count(maxWIP\ in\ WIP_{list}) = 1$ **then**
15: $command = C_{list}[indexMaxWIP]$
16: $error = WIL_{list}[indexMaxWIP]$
17: **else**
18: **if** $D_{list}[indexMinCER] = 0$ **then**
19: $command = C_{list}[indexMinCER]$
20: $error = Min(CER_{list})$
21: **else**
22: $command = Null$
23: $error = 1$
24: **end if**
25: **end if**
26: **else**
27: $trust =$ **false**
28: **if** $count(Min(WER_{list})\ in\ WER_{list}) = 1$ **then**
29: $command = C_{list}[indexMinWER]$
30: $error = Min(WER_{list})$
31: **else**
32: $command = C_{list}[indexMinCER]$
33: $error = Min(CER_{list})$
34: **end if**
35: **end if**
36: **return** $command, maxWIP, error, trust, perfect$

Source: Own authorship (2022).

Algorithm 1 estimates the command and two booleans indicating if the transcript is identical to a command or not ($perfect$) and if at least one word of the transcript matches with the estimated command ($trust$), to attend the definitions established in the component design phase. The maximum Word Information Preserved (WIP) ($maxWIP$) and the computed error ($error$) are also outputs of the command estimation function, as they may be helpful for further calculations. It needs seven lists as input, which are generated by another algorithm. The inputs are lists because they carry the values of the metrics, comparing the transcript with every possible command. The inputs are:

- Five ASR evaluation metrics, explained below, computed using the python package jiwer,

- The number of word deletions, also computed using jiwer,
- And the list of possible voice commands.

In addition to WER, presented in Subsection 3.2.3, Match Error Rate (MER), WIP, and Word Information Lost (WIL) are also calculated by the jiwer compute measures function. These three new metrics were proposed by Morris *et al.* (2004). They are calculated by Equation 8, Equation 9, and Equation 10, respectively. CER was also computed using jiwer.

All computed metrics, including WER, are based on HSDI counts at the word level. CER is also based on HSDI counts, but at the character level. H stands for hits, S for substitutions, D for deletions, and I for insertions. The Viterbi search (VITERBI, 1967) is used to align the input and output sentences, maximizing the hits, the substitutions, and finally, the deletions and insertions.

$$MER = \frac{S + D + I}{H + S + D + I} \tag{8}$$

$$WIP = \frac{H}{N_1}\frac{H}{N_2}, \tag{9}$$

where $N_1 = H + S + D$ is the total of words in the source sentence (input) and $N_2 = H + S + I$ is the total of words in the resulting transcript (output).

$$WIL = 1 - WIP \tag{10}$$

All voice commands are used as inputs to counting HSDI for estimating the most probable command, while the transcript obtained by the STT is used as output.

WIP is the first considered metric, as a WIP of 1 indicates that the sentence information was fully preserved, with the output equal to the input. If this is the case, the commands recognition subsystem finishes, calling the vehicle actuation module and acknowledging the command with this WIP as the command said by the driver.

As it is not always the case, the command with higher WIP is the most probable since $WIP > 0$ means at least one hit (a correct word). In the case of right and left turn signal commands (*seta para direita* and *seta para esquerda*, respectively), the beginning of both is the same in Portuguese (*seta para*), which can result in the same WIP, as shown in the first three examples presented in Table 8.

Table 8 – Examples of computed measures.

Example	Input	Output	WER	MER	WIP	WIL	CER
1	seta para direita	seta para direito	**0.3333**	**0.3333**	**0.4444**	**0.5556**	**0.0588**
1	seta para esquerda	seta para direito	**0.3333**	**0.3333**	**0.4444**	**0.5556**	0.3889
1	luz baixa	seta para direito	1.5	1.0	0.0	1.0	1.5556
1	pisca alerta	seta para direito	1.5	1.0	0.0	1.0	1.0833
2	seta para direita	seta para	**0.3333**	**0.3333**	**0.6667**	**0.3333**	**0.4706**
2	seta para esquerda	seta para	**0.3333**	**0.3333**	**0.6667**	**0.3333**	0.5
2	luz baixa	seta para	1.0	1.0	0.0	1.0	0.7778
2	pisca alerta	seta para	1.0	1.0	0.0	1.0	0.6667
3	seta para direita	seta	**0.6667**	**0.6667**	**0.3333**	**0.6667**	**0.7647**
3	seta para esquerda	seta	**0.6667**	**0.6667**	**0.3333**	**0.6667**	0.7778
3	luz baixa	seta	1.0	1.0	0.0	1.0	0.8889
3	pisca alerta	seta	1.0	1.0	0.0	1.0	0.6667
4	seta para direita	senta pra direi ta	**1.3333**	**1.0**	**0.0**	**1.0**	**0.1765**
4	seta para esquerda	senta pra direi ta	**1.3333**	**1.0**	**0.0**	**1.0**	0.5
4	luz baixa	senta pra direi ta	2.0	**1.0**	**0.0**	**1.0**	1.5556
4	pisca alerta	senta pra direi ta	2.0	**1.0**	**0.0**	**1.0**	1.0

Source: Own authorship (2022).

In examples 1, 2, and 3, the commands *seta para direita* and *seta para esquerda* have the same WIP. The same happens to the other word level measures (WER, MER, and WIL). Both commands have the same number of hits (shown in Table 9) in the three examples. Therefore, CER is the only different metric between both commands, being lower in the three examples in *seta para direita*. In the first example, we can assume that *seta para direita* is the most probable command, but we cannot suppose the same for examples 2 and 3 due to the deletions shown in Table 9. This way, if more than one command has the same WIP value, with $WIP > 0$, the command with the lowest CER is chosen as long as it has no deletions. If there are deletions, no command is chosen by the STT module.

Table 9 – Word level HSDI counts example.

Input	Output	Hits	Substitutions	Deletions	Insertions
seta para direita	seta para direito	2	1	0	0
seta para esquerda	seta para direito	2	1	0	0
seta para direita	seta para	2	0	1	0
seta para esquerda	seta para	2	0	1	0
seta para direita	seta	1	0	2	0
seta para esquerda	seta	1	0	2	0

Source: Own authorship (2022).

In case of no hits at the word level, MER is always 1, WIP is 0, and WIL is 1, which means that only WER will vary, assuming values greater than or equal to 1, only depending on the number of insertions, as shown in example 4 in Table 8. In this particular situation, the command with lower WER is the most probable if there is a single command with this WER, but

it is not reliable. If there is more than one command with the lowest WER, the command with the lower CER is the most probable, but it is unreliable as well.

Algorithm 2 computes the ASR metrics for each command to generate the lists needed by Algorithm 1 to estimate the voice command. It uses the transcript obtained from the STT model and the list of possible voice commands.

Algorithm 2 – Compute ASR measures.

Input: t (transcript), C_{list} (list of commands)
1: **for each** $c \in C_{list}$ **do**
2: H_w,S_w,D_w,$I_w = CountWordHSDI(c,t)$
3: H_c,S_c,D_c,$I_c = CountCharacterHSDI(c,t)$
4: $WER = (S_w + D_w + I_w)/(H_w + S_w + D_w)$
5: $CER = (S_c + D_c + I_c)/(H_c + S_c + D_c)$
6: $MER = (S_w + D_w + I_w)/(H_w + S_w + D_w + I_w)$
7: $WIP = (H_w/(H_w + S_w + D_w)) * (H_w/(H_w + S_w + I_w))$
8: $WIL = 1 - WIP$
9: append WER to WER_{list}
10: append CER to CER_{list}
11: append MER to MER_{list}
12: append WIP to WIP_{list}
13: append WIL to WIL_{list}
14: append D_w to D_{list}
15: **end for**
16: **return** $WER_{list}, CER_{list}, MER_{list}, WIP_{list}, WIL_{list}, D_{list}$

Source: Own authorship (2022).

3.2.6.3.2 Voice features similarity

This module calculates the similarity of the recorded voice command embedding to each command *.npy* file created in the driver enrollment. It receives the transcript obtained by the STT module, the list of commands and their embeddings, and the audio embedding. The similarity of the audio embedding with each command embedding is calculated using Resemblyzer, as shown in Algorithm 3. It defines the command with the higher similarity as the most probable command ($command$). Its similarity ($score2$) and the similarity of the STT command ($score1$) are also outputs from the module, as specified in the components design phase.

Algorithm 3 – Voice features similarity.

Input: t (transcript), C_{list} (list of commands), E_{list} (list of commands embeddings), A (audio embedding)

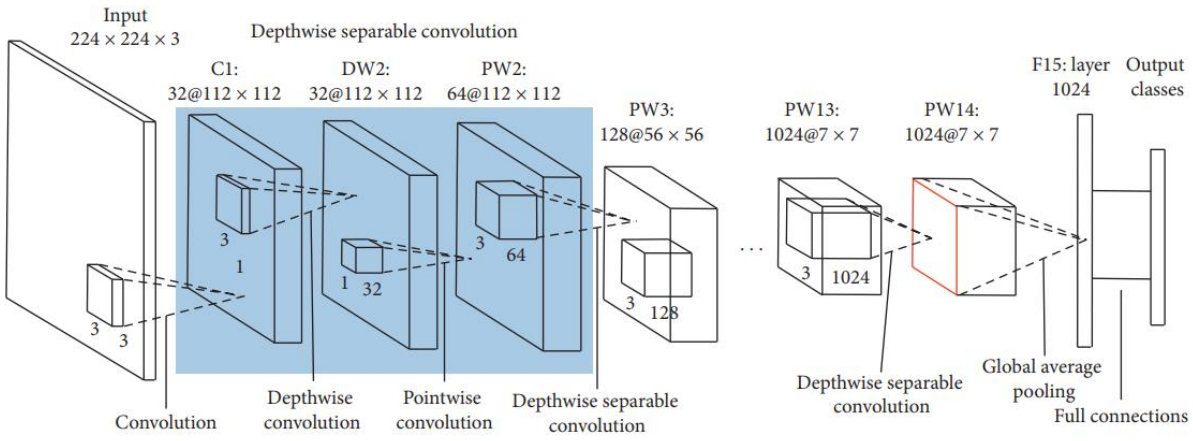
$maxSimilarity = 0$
for $index = 0$ **to** $Lenght(C_{list})$ **do**
  $similarity = CalcSimilarity(A, E_{list}[index])$
  $cmd = C_{list}[index]$
  if $t = cmd$ **then**
    $score1 = similarity$
  end if
  if $similarity > maxSimilarity$ **then**
    $score2 = similarity$
    $maxSimilarity = similarity$
    $command = cmd$
  end if
end for
return $command$, $score1$, $score2$

Source: Own authorship (2022).

3.2.6.3.3 Spectrogram classification

This module uses the best spectrogram classifier model, selected based on the results presented in Section 4.4. Figure 24 illustrates the MobileNet architecture as it was the chosen model.

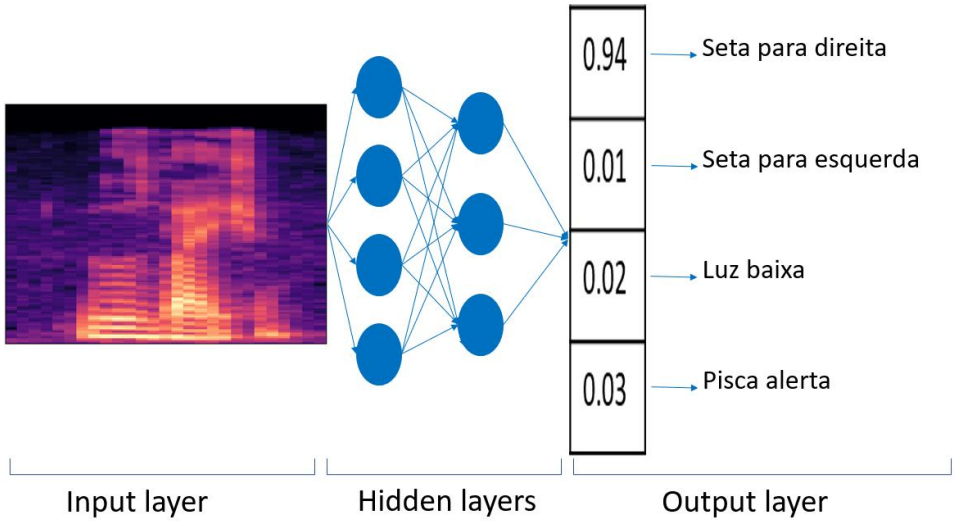
Figure 24 – MobileNet classifier model architecture.



Source: Wang *et al.* (2020).

The model predicts a class to the input spectrogram image, which, in this case, is one of the possible voice commands. Every classifier model that was trained has four possibilities in its output layer, corresponding to the four voice commands used in this work, as shown in Figure 25, which presents an example of a generic classifier model with a spectrogram in the input layer, the hidden layers, and the output layer, predicting *seta para direita* as the most probable command with a probability of 94%.

Figure 25 – Generic spectrogram classifier model architecture.



Source: Own authorship (2022).

Each class has a probability calculated by the model, and the class with the highest probability is selected as the most probable command, which is one output of the spectrogram classification module with its associated score. It concludes the commands recognition subsystem.

3.2.6.3.4 Vehicle actuation

The vehicle actuation module receives data from the outputs of the three modules from the commands recognition subsystem. It is called after the three-step verification or right after the STT if it returns a perfect transcript, as explained in Subsection 3.2.6.3.1.

We applied the logic described in Algorithm 4 to decide which command should be activated. C_1 is the command from STT, C_2 is the command from voice features similarity, C_3 is the command from spectrogram classification, and $prob$ is its probability calculated by the classifier. $perfect$ and $trust$ are outputs from Algorithm 1, and $score1$ and $score2$ are outputs from Algorithm 3. $useC_2$ and $useC_3$ are booleans to indicate if the second verification (voice features similarity) and third verification (spectrogram classification) should be used. Thresholds 1 and 2 will be defined later, based on tests comparing voice features (Subsection 4.5.3). The threshold 3 will be defined based on tests of the spectrogram classifier model (Section 4.6). Algorithms 5 and 6 show the auxiliary functions used in Algorithm 4.

Algorithm 4 – Decision of which command should be actuated.

Input: C_1,$perfect$,$trust$, C_2, $score1$, $score2$,$threshold1$,$threshold2$,C_3,$prob$,$threshold3$,$useC_2$,$useC_3$,C_{list}

if $perfect$ **then**
  $command = C_1$
else if $C_1 = Null$ **then**
  if $C_2 = C_3$ **then**
    $command = C_2$
  else
    $command = Null$
  end if
else
  if $trust$ **then**
    if $useC_2$ **then**
      if $useC_3$ **and** $prob \geq threshold3$ **then**
        $command = ThreeVerification(C_1, C_2, score1, score2, threshold1, C_3, C_{list})$
      else
        $command = TwoVerification(C_1, C_2, score1, score2, threshold1, threshold2)$
      end if
    else
      $command = C_1$
    end if
  else
    $command = Null$
  end if
end if
return $command$

Source: Own authorship (2022).

Algorithm 5 – Decision of which command should be actuated: Auxiliary function $ThreeVerification()$**.**

Input: C_1, C_2, $score1$, $score2$, $threshold1$, C_3, C_{list}

$maxCount = 0$
for each $c \in C_{list}$ **do**
  $count = 0$
  if $c = C_1$ & $score1 \geq threshold1$ **then**
    $count++$
  else if $c = C_2$ & $score2 \geq threshold1$ **then**
    $count++$
  else if $c = C_3$ **then**
    $count++$
  end if
  if $count > maxCount$ **then**
    $maxCount = count$
    $tempCommand = c$
  end if
end for
if $maxCount > 1$ **then**
  $command = tempCommand$
else
  $command = Null$
end if
return $command$

Source: Own authorship (2022).

Algorithm 6 – Decision of which command should be actuated: Auxiliary function $TwoVerification()$.

Input: C_1, C_2, $score1$, $score2$, $threshold1$,$threshold2$
if $C_1 = C_2$ **then**
  $command = C_1$
else
  if $score1 \geq threshold1$ **then**
    if $score2 \geq threshold1$ **then**
      if $scorer1/scorer2 \geq threshold2$ **then**
        $command = C_1$
      else
        $command = C_2$
      end if
    else
      $command = C_1$
    end if
  else
    if $score2 \geq threshold1$ **then**
      $command = C_2$
    else
      $command = Null$
    end if
  end if
end if
return $command$

Source: Own authorship (2022).

This module needs to check if the command is activated (or not) to give the proper feedback to the user. If it is activated, it should deactivate and vice-versa, announcing the action to the user with the driver report module. The module realizes this check after deciding on the command to be actuated.

The vehicle actuation needs to be simulated in the computer for the first version of the system, as defined in Subsection 3.1.2 (system design). That way, we used an array of logical values for simulating the vehicle systems, as shown in Equation 11. A True logical value indicates that the system is already activated.

$$vehicleSignals = \begin{bmatrix} rightTurn & leftTurn & hazardWarning & headlights \end{bmatrix} \quad (11)$$

If the left turn signal is active and the right turn command is received, the left turn signal is deactivated before activating the right turn signal. The same happens to the right turn signal if the left turn command is received.

3.2.6.3.5 Storage of new features

After actuating, the audio features (embedding and spectrogram) can be saved. The spectrograms can be used for training new models aiming for further improvements. In this case, the embeddings used for speaker verification and by the Voice Features Similarity module of command recognition are updated. The consequences of these updates need to be analyzed, and this module is disabled by default.

It receives the command chosen by the vehicle actuation module, the driver folder path, the spectrogram, and the embedding array. The embedding array is appended to the driver array (*driver.npy*) and the *.npy* array relative to the proper command. The spectrogram is saved to the command folder.

4 RESULTS AND DISCUSSION

This Chapter presents the procedures and results of the experiments and the initial simulation results.

4.1 Simulation results

4.1.1 Simulink requirements and tests

Figure 26 shows the Simulink Requirements Editor tool after simulating. All tests (system tests, integration tests, and component tests) were implemented and verified.

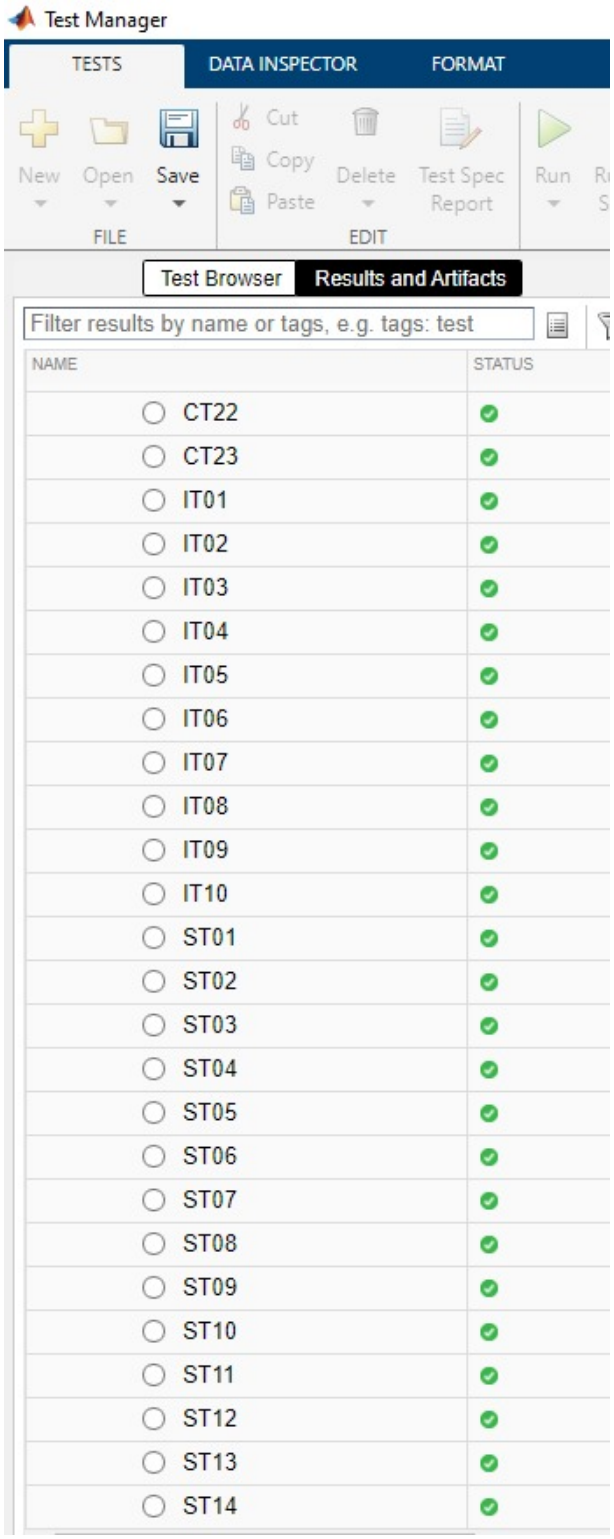
Figure 26 – Simulink Requirements Editor.

Index	ID	Summary	Implemented	Verified
VoiceAssistant...				
1	R01	The system must start receiving voice commands upon request		
1.1	R01.1	The system must have a microphone		
1.2	R01.2	The system must have a button		
1.3	R01.3	The system must record audio only when required		
2	R02	The system must indicate when it is ready to receive the voice commands		
2.1	R02.1	The system must have speakers		
2.2	R02.2	The system must be able to speak and emit sounds warnings		
2.3	R02.3	The system must instruct the user on how to use the system		
3	R03	The system must be able to verify if the voice command is from the current driver		
3.1	R03.1	The system must verify if the recorded audio has voice		
3.1.1	R03.1.1	The system must report when no speech is detected		
3.2	R03.2	The system must identify the driver through its voice		
3.2.1	R03.2.1	The system must report when the driver is not identified		
3.2.2	R03.2.2	The system must report when it identifies the driver		
3.3	R03.3	The system must check if the recorded voice is from the driver		
3.3.1	R03.3.1	The system must report when the voice fails the verification		
3.4	R03.4	The system must store data from new driver enrollments		
3.4.1	R03.4.1	The system must be able to store new data from the driver		
4	R04	The system must respond to specific voice commands		
4.1	R04.1	The system must recognize the voice commands		
4.1.1	R04.1.1	The system must report if a voice command was not recognized		
4.2	R04.2	The system must not act if the command is not from the driver		
4.3	R04.3	The system must actuate on the vehicle if the command was recognized		
4.3.1	R04.3.1	The system must report when a command was received		
4.3.2	R04.3.2	The system must check the current status of the vehicle's dashboard before acting		

Source: Own authorship (2022).

Figure 27 shows the Simulink Test Manager Results screen, with all tests verified.

Figure 27 – Simulink Test Manager.



Source: Own authorship (2022).

After the simulations, the Test Manager tool allows the user to generate an automatic report of the tests, which is partially presented in Figure 28.

Figure 28 – Simulink tests automatic report.

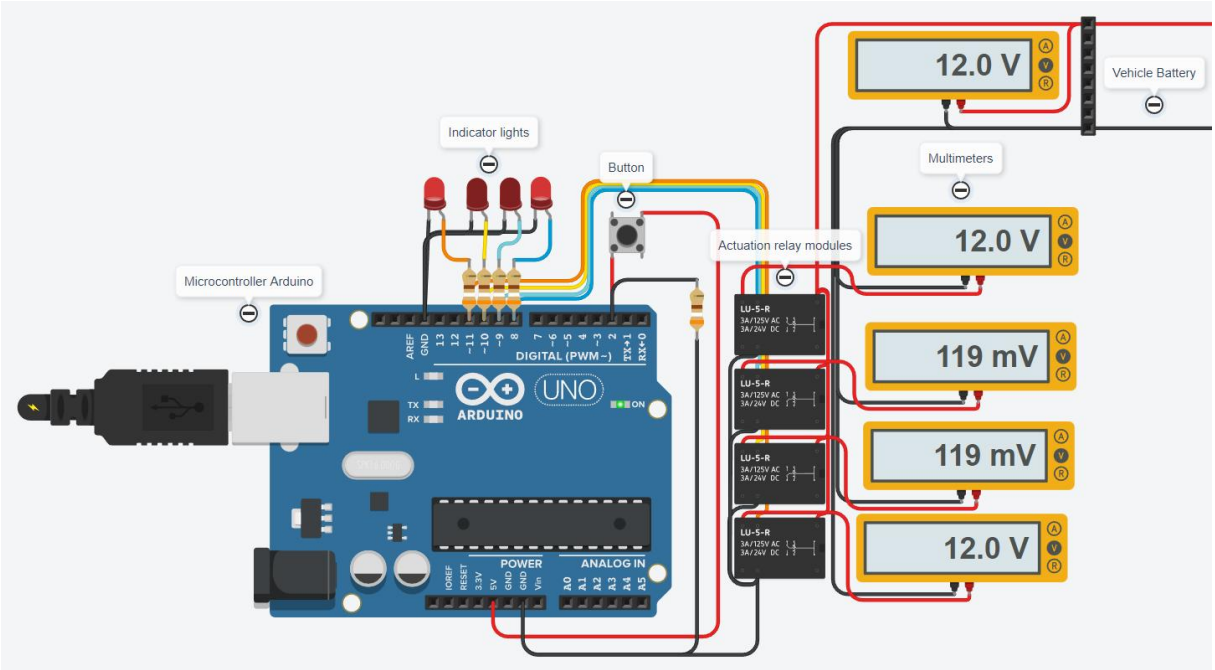
ST01	At any point in time, if (State == 4) becomes true then, with no delay, (Indication == 1) must be true **REQUIREMENTS** **Description:** R02 The system must indicate when it is ready to receive the voice commands **Document:** VoiceAssistantSystemRequirements.slreqx
ST02	At any point in time, whenever (ButtonPressed ~= 1) is true then, with no delay, (Recording == 0) must be true **REQUIREMENTS** **Description:** R01.3 The system must record audio only when required **Document:** VoiceAssistantSystemRequirements.slreqx

Source: Own authorship (2022).

4.1.2 Hardware simulation

The hardware architecture of the second version of the prototype was simulated before the implementation, as shown in Figure 29. The simulation was made in Autodesk Tinkercad, using an Arduino Uno as the microcontroller.

Figure 29 – System prototype (V2) simulation.



Source: Own authorship (2022).

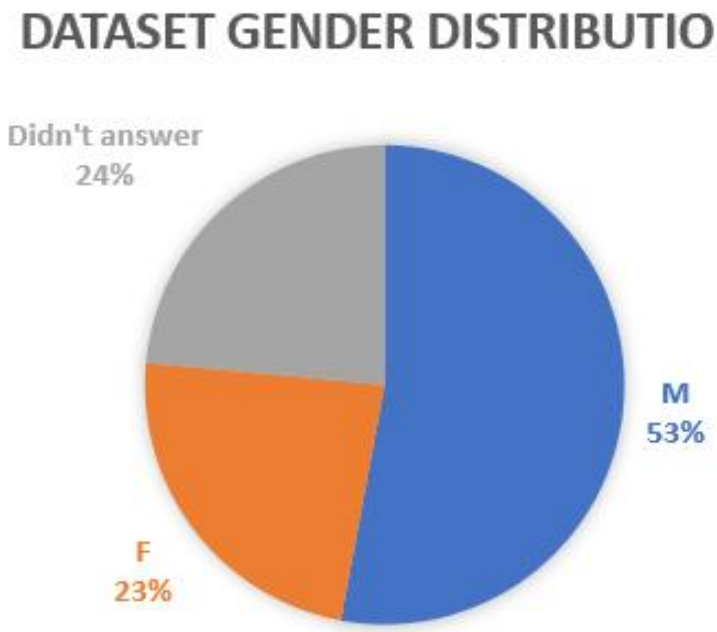
The communication between the system and the microcontroller was simulated by sending messages through the Arduino Serial Monitor. In addition to the hardware defined in the requirements table (Frame 1), four LEDs were used to indicate which signal was activated in

conjunction with the relay modules. The simulation also had multimeters to measure the voltage level of the simulated vehicle battery and each relay.

4.2 Dataset creation

A total of 17 people participated in the creation of the audio database, of which six did not answer their age, and the average age of the other participants was around 26. The gender distribution is presented in Graph 2.

Graph 2 – Gender distribution of participants in the audio dataset creation.



Source: Own authorship (2022).

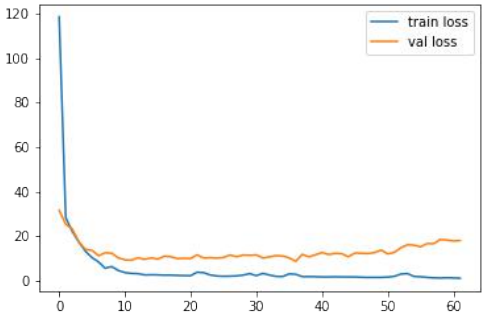
A total of 204 *.wav* files were recorded, all with a duration of 4 seconds.

In the expanded dataset VVCMD, 192 synthesized audios, obtained using Microsoft Azure TTS, were added to the original dataset, totaling 396 audios of voice commands. The artificial audios do not have the same duration since they were recorded and cut using audio editing software.

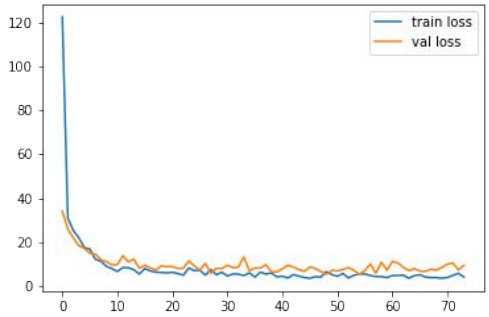
4.3 Speech-to-Text acoustic and language models

Graph 3 shows the logs of training and validation losses from all four defined configurations for AM model training in the VVCD dataset.

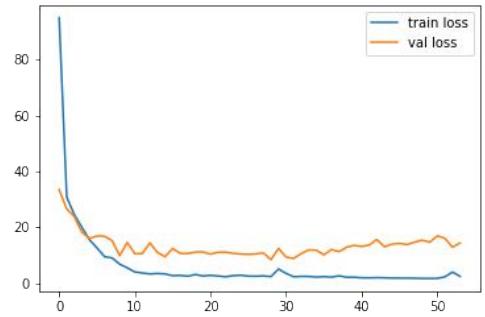
Graph 3 – Logs from training and validation loss from acoustic models training. The x-axis represents the number of training epochs and the y-axis represents the loss value.



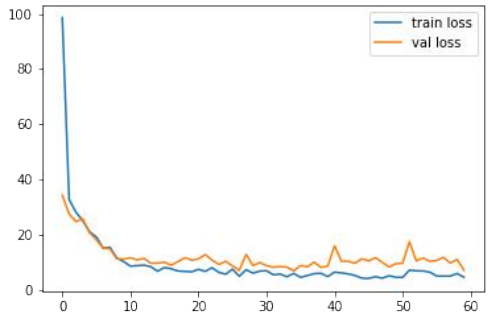
(a) Without data augmentation



(b) With data augmentation



(c) Without data augmentation with 20% dropout



(d) With data augmentation with 20% dropout

Source: Own authorship (2022).

Overfitting can be noticed in both training sessions without data augmentation since the validation loss started increasing after some epochs while the training loss continued decreasing, but this is less noticeable in the case with higher dropout. In both cases, early stopping helped to speed up the training process by stopping the training when it was no longer improving the validation results. It also reduced the overfitting effect.

Table 10 shows information obtained during the language model training with each dataset. The top k-words percent calculates the percentage of the whole corpus of the words that appear the most. In this case, the 50000 most common words correspond to more than 90% in all three corpora. The most common word of all datasets was *"de"*, a Portuguese preposition.

Table 10 – Results of the language model training.

Dataset	Total words	Unique words	Most common word	Top k-words percent
MLS	13837584	383198	de	94.109%
Europarl	49937799	402469	de	97.8%
WikiText PT-BR	194708378	1164234	de	96.008%

Source: Own authorship (2022).

The MLS dataset, which has less than 14 million words, is the smallest, while WikiText

PT-BR, the largest, has more than 194 million words, 14 times the size of MLS. Each corpus received one million words resultant from the addition of 100000 sentences of each command, which is more significant for MLS for being the smaller and less impactful to a larger corpus like WikiText PT-BR, making its probabilities the lowest.

Table 11 shows the quantity and probability of each word to happen in the training corpora after the addition, with the highest probability highlighted in bold. Since the highest probabilities were all in the modified MLS, it was used to train four new language models, considering a 5-gram, a 4-gram, a trigram, and a bigram. For the latter, singleton bigrams were pruned, while singleton trigrams and above were pruned for the others.

Table 11 – Word count in each modified corpus.

Word	MLS	Europarl	WikiText PT-BR
Alerta	**100021 (0.674%)**	100951 (0.198%)	102187 (0.052%)
Baixa	**100802 (0.679%)**	101145 (0.199%)	122251 (0.062%)
Direita	**101256 (0.682%)**	100863 (0.198%)	112291 (0.057%)
Esquerda	**100912 (0.680%)**	101003 (0.198%)	114740 (0.059%)
Luz	**105637 (0.712%)**	104095 (0.204%)	126075 (0.064%)
Para	**304982 (2.055%)**	731769 (1.437%)	1882283 (0.962%)
Pisca	**100010 (0.674%)**	100000 (0.196%)	100110 (0.051%)
Seta	**200061 (1.348%)**	200001 (0.393%)	200449 (0.102%)

Source: Own authorship (2022).

Table 12 shows the performance (average of WER and CER) of each acoustic model in the test set of VVCD, scored with each language model. A total of 31 audio files were tested, corresponding to 15% of the dataset.

Table 12 – Different training configurations results on test dataset.

AM	LM	WER	CER
1	MLS	0.3226	0.1070
2	MLS	**0.3172**	0.1091
3	MLS	0.4086	0.1437
4	MLS	0.3710	0.1300
1	Europarl	0.3656	0.1335
2	Europarl	0.3817	0.1398
3	Europarl	0.3656	0.1335
4	Europarl	0.3656	0.1158
1	WikiText PT-BR	0.3656	0.1237
2	WikiText PT-BR	0.3656	0.1276
3	WikiText PT-BR	0.3602	0.1419
4	WikiText PT-BR	0.3387	**0.1050**

Source: Own authorship (2022).

The second configuration of AM, which used 5% dropout and data augmentation, achieved the best WER when evaluated by the LM trained with the MLS corpus. It got a WER

of 0.3172, followed by the first AM with MLS LM and the fourth AM with the WikiText PT-BR LM, which had the best CER.

All acoustic models presented better results when evaluated by the LM trained with the modified MLS, as shown by the average WER and CER obtained by each AM, presented in Table 13.

Table 13 – Word error rate and character error rate in different n-grams trained with modified MLS dataset.

Metric	AM	5-gram	4-gram	Trigram	Bigram
WER	1	0.0484	0.0484	0.0484	**0.0376**
WER	2	0.0376	0.0376	0.0376	**0.0269**
WER	3	**0.0323**	**0.0323**	**0.0323**	**0.0323**
WER	4	0.0323	0.0323	0.0323	**0.0215**
CER	1	0.0221	0.0221	0.0221	**0.0202**
CER	2	0.0218	0.0218	0.0218	**0.0199**
CER	3	**0.0161**	**0.0161**	**0.0161**	**0.0161**
CER	4	0.0114	0.0114	0.0114	**0.0095**

Source: Own authorship (2022).

A nearly ten times improvement in both metrics is perceived when comparing the 5-gram model with the results of Table 12, which also used 5-gram language models. The best results of all AM were obtained using the bigram language model, with a highlight on the fourth AM, which had the lowest WER (0.0215) and CER (0.0095). The other AM that used data augmentation (model 2) had the second-best WER.

WikiText PT-BR dataset is the largest, and its resulting LM obtained the third best WER when combined with the fourth AM in Table 12. This combination also got the best CER. That way, the modified WikiText PT-BR dataset, after adding the 100000 sentences of each voice command, was also used to train four new language models, considering the 5-gram, 4-gram, trigram, and bigram. The average WER and CER are shown in Table 14.

Table 14 – Word error rate and character error rate in different n-grams trained with modified WikiText PT-BR dataset.

Metric	AM	5-gram	4-gram	Trigram	Bigram
WER	1	**0.0484**	**0.0484**	**0.0484**	**0.0484**
WER	2	**0.0269**	**0.0269**	**0.0269**	0.0376
WER	3	**0.0323**	**0.0323**	**0.0323**	**0.0323**
WER	4	**0.0376**	**0.0376**	0.0484	**0.0376**
CER	1	**0.0221**	**0.0221**	**0.0221**	0.0256
CER	2	**0.0145**	**0.0145**	**0.0145**	0.0183
CER	3	**0.0161**	**0.0161**	**0.0161**	**0.0161**
CER	4	**0.0202**	**0.0202**	0.0221	**0.0202**

Source: Own authorship (2022).

In this case, all acoustic models had the same results in at least three n-grams, and AM 1 got the same WER with all n-grams. The same happened to AM 3, which had the same CER

with all n-grams.

The best WER (0.0269) was obtained by AM 2 when scored by the 5-gram, 4-gram, and trigram language models. This model also got the best CER (0.01454). But even this model was not able to overcome the results of the AM 4 combined with the bigram LM trained using the modified MLS dataset.

The acoustic models which used data augmentation for training got the best results in all evaluated cases (from tables 12 to 14). Due to this, we trained two new acoustic models using data augmentation and ranging the dropout rate from 5% (which will be called M1) to 20% (which will be called M2) using the VVCMD.

Due to the improvements of adding the 100000 sentences of every voice command to the modified MLS and WikiText PT-BR datasets, their resulting language models were used to score the new acoustic models M1 and M2. The average WER and CER are shown in Table 15. These results are relative to the test set of the VVCMD, which had 60 audios.

Table 15 – Word error rate and character error rate in different n-grams trained with modified MLS and WikiText PT-BR datasets and acoustic models trained with VVCMD.

Metric	AM	LM dataset	5-gram	4-gram	Trigram	Bigram
WER	M1	MLS	0.5278	0.5278	0.5306	**0.5139**
WER	M2	MLS	0.4389	0.4389	0.4250	**0.4167**
WER	M1	WikiText PT-BR	**0.5111**	**0.5111**	0.5472	0.5139
WER	M2	WikiText PT-BR	**0.4111**	0.4222	0.4167	0.4167
CER	M1	MLS	0.3767	0.3767	0.3804	**0.3761**
CER	M2	MLS	0.2905	0.2925	0.2911	**0.2845**
CER	M1	WikiText PT-BR	0.3917	0.3917	0.3908	**0.3776**
CER	M2	WikiText PT-BR	**0.2943**	0.2962	0.3056	0.2974

Source: Own authorship (2022).

Once again, the model trained with the MLS dataset had the best results in the bigram. In both datasets used for LM training, the AM trained with 20% dropout excelled. The best WER (0.4111) was obtained by AM M2 with the 5-gram LM trained with the WikiText PT-BR dataset. The best CER (0.2845) was obtained by AM M2 with the bigram LM trained with the MLS dataset.

Table 16 shows the file size of each generated language model, comparing the size of the n-grams models trained using the modified MLS and modified WikiText PT-BR datasets.

Table 16 – File size (MB) of the different n-grams trained with modified MLS and WikiText PT-BR datasets.

Dataset	5-gram	4-gram	Trigram	Bigram
MLS	22.7	21.0	17.5	**4.78**
WikiText PT-BR	201.0	155.0	97.1	**16.4**

Source: Own authorship (2022).

Besides being the smallest model, the bigram trained with the modified MLS dataset got the best results in the VVCD, as shown in Table 13. It also got very close results to the biggest model, the 5-gram trained with WikiText PT-BR, on the VVCMD with a WER of 0.4167 compared to 0.4111, which is quite impressive considering it has only 2.4% of the size of its concurrent. That way, the bigram language model trained with the modified MLS was selected to be used in the STT module of the commands recognition subsystem, together with the acoustic model M2, which was trained using the VVCMD with data augmentation and the dropout rate of 20%.

4.4 Spectrogram classifier

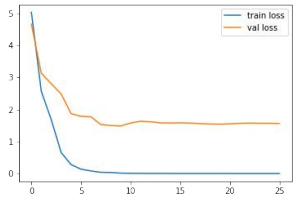
The spectrogram classifier models were trained using transfer learning on 15 different Keras models for image classification, aiming to predict what command has been said through the spectrogram of the recorded audio. All the outcomes below are from training sessions that used early stopping based on validation accuracy because those parameters produced the best results.

Graph 4 shows the plottings of the calculated training and validation losses. Graph 5 shows the accuracy logs.

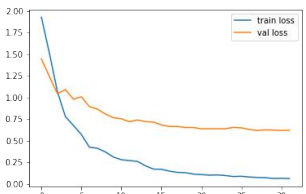
Underfitting can be noticed in the results of EfficientNetB7 by its validation accuracy in Graph 5(o). It had a high loss in training and validation, as seen in Graph 4(o). Consequently this model had the worst accuracy in training (0.2929) and validation (0.25), as presented in Table 17.

Graph 6 shows the confusion matrix of each model, obtained after training, with the test dataset. It is remarkable again that the EfficientNetB7 model did not do very well, assuming that all images belonged to the same (*seta_direita*), indicating that it was unable to understand the relationship between the input images and the output classes.

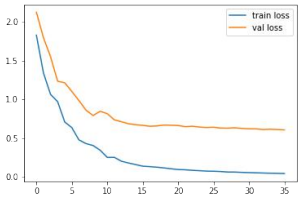
Graph 4 – Spectrogram classifier training log: training and validation losses. The x-axis represents the number of training epochs and the y-axis represents the loss value.



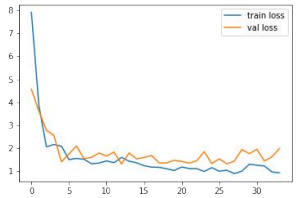
(a) Xception



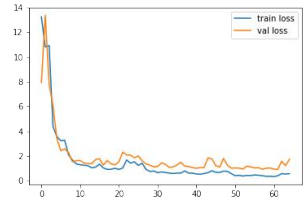
(b) VGG16



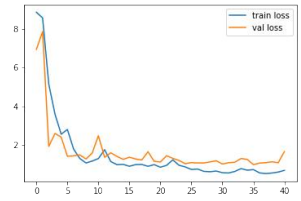
(c) VGG19



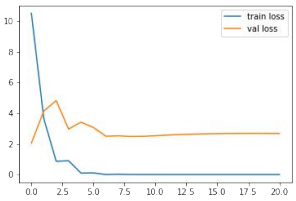
(d) ResNet50



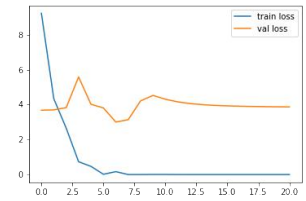
(e) ResNet101



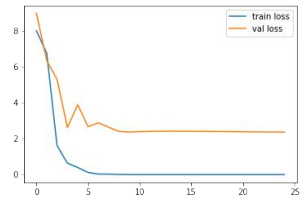
(f) ResNet152



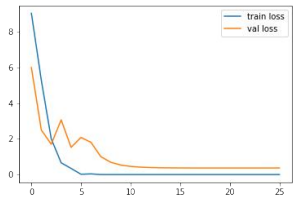
(g) ResNet50V2



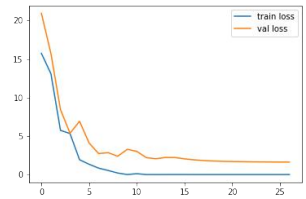
(h) ResNet101V2



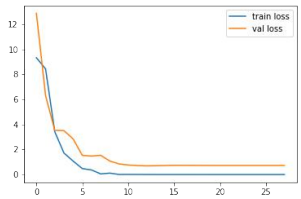
(i) ResNet152V2



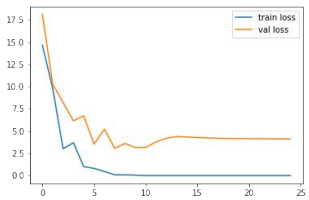
(j) MobileNet



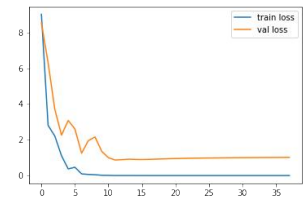
(k) InceptionV3



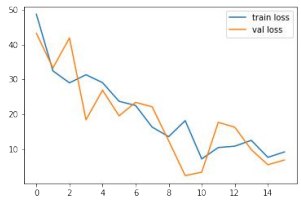
(l) InceptionResNetV2



(m) MobileNetV2



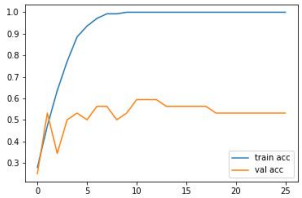
(n) DenseNet201



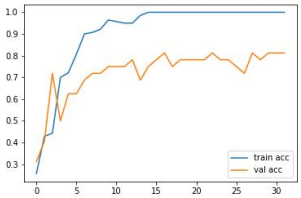
(o) EfficientNetB7

Source: Own authorship (2022)

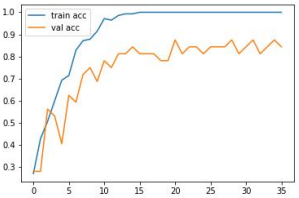
Graph 5 – Spectrogram classifier training log: training and validation accuracies. The x-axis represents the number of training epochs and the y-axis represents the accuracy value.



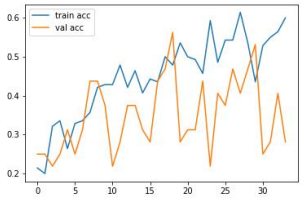
(a) Xception



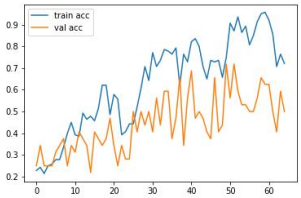
(b) VGG16



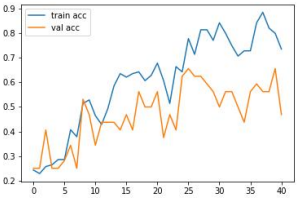
(c) VGG19



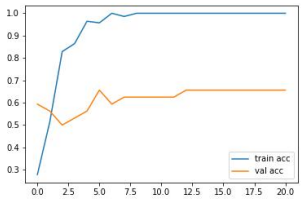
(d) ResNet50



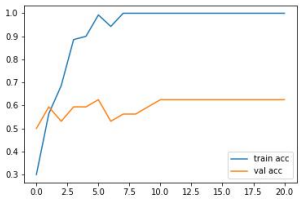
(e) ResNet101



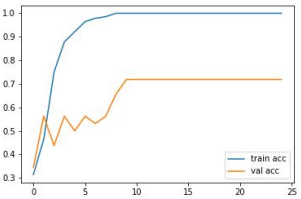
(f) ResNet152



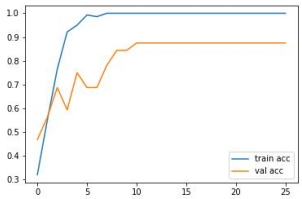
(g) ResNet50V2



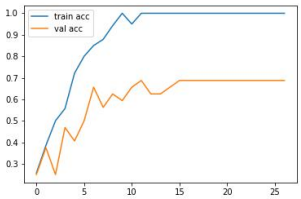
(h) ResNet101V2



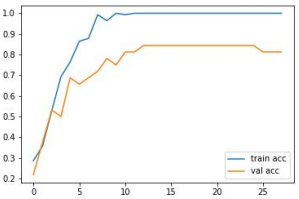
(i) ResNet152V2



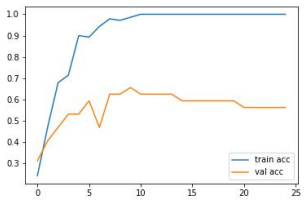
(j) MobileNet



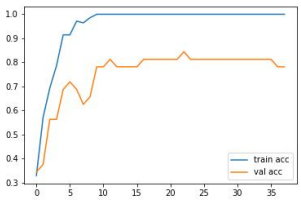
(k) InceptionV3



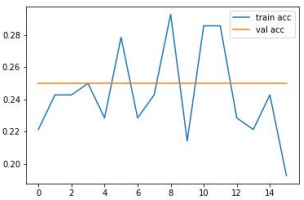
(l) InceptionResNetV2



(m) MobileNetV2



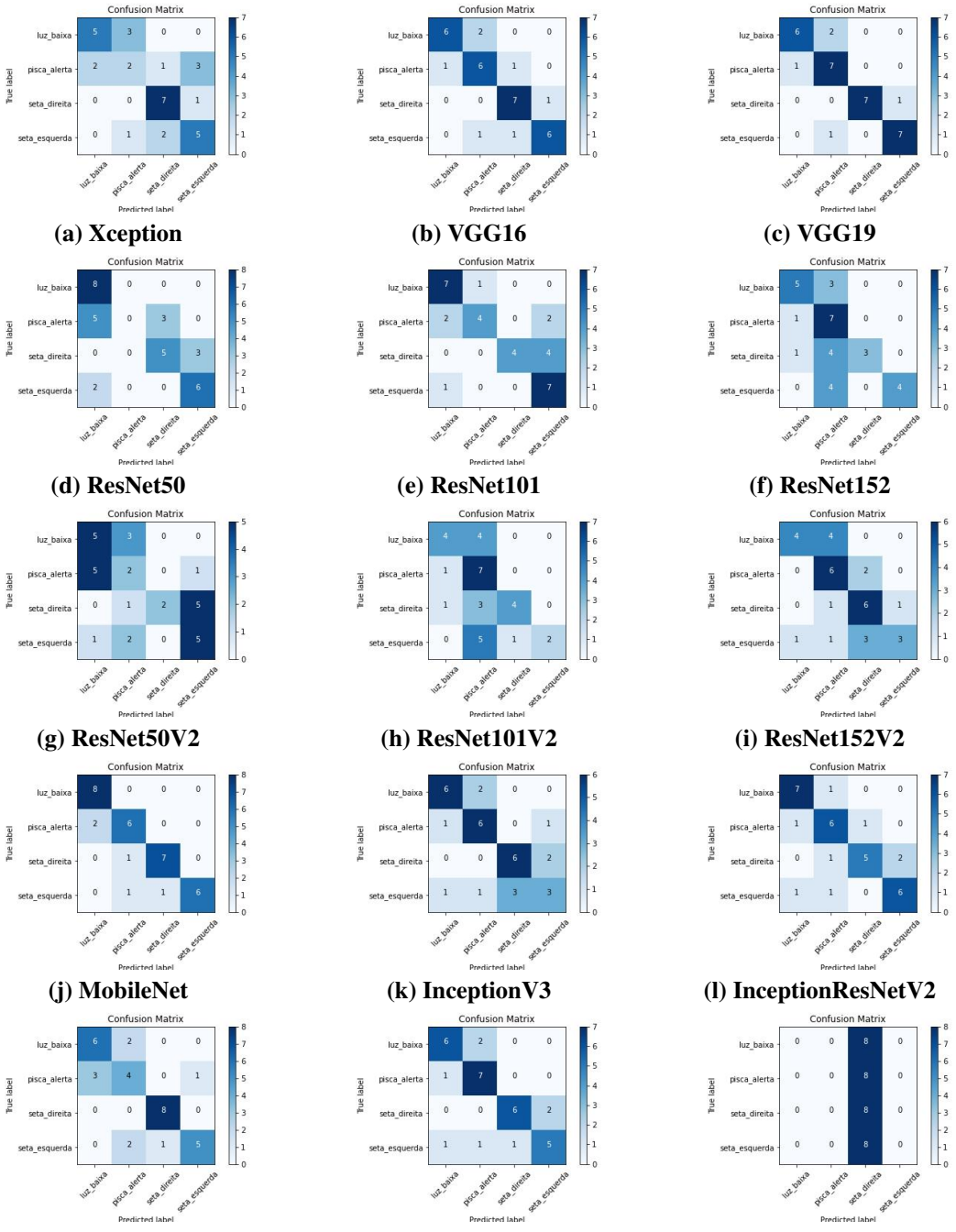
(n) DenseNet201



(o) EfficientNetB7

Source: Own authorship (2022)

Graph 6 – Spectrogram classifier training results: confusion matrices.



(a) Xception **(b) VGG16** **(c) VGG19**

(d) ResNet50 **(e) ResNet101** **(f) ResNet152**

(g) ResNet50V2 **(h) ResNet101V2** **(i) ResNet152V2**

(j) MobileNet **(k) InceptionV3** **(l) InceptionResNetV2**

(m) MobileNetV2 **(n) DenseNet201** **(o) EfficientNetB7**

Source: Own authorship (2022)

Table 17 shows the best loss and accuracy achieved during the training and validation of each model. Eleven models obtained the maximum accuracy during training, but some were not as good in the validation, indicating that overfitting may have occurred. VGG16 and VGG19 stood out for their small validation and losses, respectively. VGG19 had the best validation accuracy of 0.875, tied with MobileNet, followed by DenseNet201 and InceptionResNetV2, both with 0.8437, and VGG16, in third place, with 0.8125.

Table 17 – Spectrogram classifiers training and validation best losses and accuracies.

Model name	Best training loss	Best validation loss	Best training accuracy	Best validation accuracy
DenseNet201	9.0198	8.5428	**1.0**	0.8437
EfficientNetB7	48.7413	43.2852	0.2929	0.25
InceptionResNetV2	9.3369	12.8878	**1.0**	0.8437
InceptionV3	15.7487	20.9589	**1.0**	0.6875
MobileNet	9.0384	5.9986	**1.0**	**0.875**
MobileNetV2	14.6501	18.1384	**1.0**	0.6562
ResNet101	13.2373	13.3655	0.9571	0.7187
ResNet101V2	9.2219	5.581	**1.0**	0.625
ResNet152	8.8628	7.8643	0.8857	0.6562
ResNet152V2	8.0301	9.0044	**1.0**	0.7187
ResNet50	7.9151	4.5675	0.6143	0.5625
ResNet50V2	10.5207	4.8236	**1.0**	0.6562
VGG16	1.93	**1.4477**	**1.0**	0.8125
VGG19	**1.831**	2.1257	**1.0**	**0.875**
Xception	5.0333	4.6662	**1.0**	0.5937

Source: Own authorship (2022).

The Table 18 shows the accuracy that each model achieved during the test step of the training process and the file size of each resulting model. MobileNet and VGG19 stood out for their accuracy (0.844), but MobileNet has the advantage due to its size (corresponding to 19% of the size of the VGG19 model), which can be crucial for embedded processes. The best accuracy obtained in the training that used validation loss as the early stopping condition was also from VGG19, with 0.844, which was the same value obtained in the first training. The smallest model was MobileNetV2, with only 11.9 MB and an accuracy of 0.719, which is better than most models. Despite being the largest model, EfficientNetB7 achieved the worst accuracy, only 0.25, meaning that the model could not learn during training which is quite noticeable from the confusion matrix presented in Graph 6(o), in which the predicted class is always the same. All these considerations led to the selection of MobileNet for the spectrogram classification module.

Table 18 – Spectrogram classifiers test accuracies and models file size.

Model name	Accuracy	Size (MB)
DenseNet201	0.750	76.0
EfficientNetB7	0.250	252.0
InceptionResNetV2	0.750	211.0
InceptionV3	0.656	86.3
MobileNet	**0.844**	14.8
MobileNetV2	0.719	**11.9**
ResNet101	0.688	168.0
ResNet101V2	0.531	168.0
ResNet152	0.594	228.0
ResNet152V2	0.594	228.0
ResNet50	0.594	95.0
ResNet50V2	0.438	95.0
VGG16	0.781	57.3
VGG19	**0.844**	77.6
Xception	0.594	84.5

Source: Own authorship (2022).

4.5 Voice features similarities

The comparison of voice features is used by three modules of the system: driver identification, speaker verification, and voice features similarity.

For each speaker of the VVCD, two audios of every voice command were selected to calculate the similarity of the possible pairs. A total of 16 pairs were formed per speaker. Each audio was used to create four pairs, totaling 136 cases (17 speakers $\times$ 8 audios per speaker) to be analyzed.

The higher similarity of each line of Table 19 and Table 20 is highlighted in bold. Table 19 shows the average similarity of all pairs. The average similarity of an AA pair (formed by two audios of the same command) is higher than the AB pairs (formed by two audios of different commands).

Table 19 – Average similarities between commands of the same speaker.

Command	Headlights	Hazard Warning	Right Turn Signal	Left Turn Signal
Headlights	**0.8799**	0.7583	0.7108	0.7069
Hazard Warning	0.7775	**0.8918**	0.7647	0.8087
Right Turn Signal	0.7031	0.7470	**0.8727**	0.8048
Left Turn Signal	0.7504	0.8038	0.8543	**0.9026**

Source: Own authorship (2022).

Each line in Table 20 shows the minimum similarity obtained by each AA pair. When comparing one audio of headlights (A1) to another audio of headlights (A2), one of the speakers got a similarity of 0.7536, which is lower than the similarity of A1 compared to one hazard

warning audio (0.7782). According to the results, an AA pair can have less similarity than an AB pair. That happened 11 times out of 136 cases, which means that in 86% of the cases, the similarity of an audio is higher when compared to another audio of the same speaker saying the same command.

Table 20 – Minimum similarities in AA pairs of the same speaker.

Command	Headlights	Hazard Warning	Right Turn Signal	Left Turn Signal
Headlights	0.7536	**0.7782**	0.6653	0.6546
Hazard Warning	**0.8593**	0.8271	0.5890	0.7834
Right Turn Signal	0.7417	0.7824	0.7555	**0.8575**
Left Turn Signal	0.7248	0.8119	**0.8795**	0.7977

Source: Own authorship (2022).

The 16 Microsoft Azure speakers were subjected to the same analysis. The similarities between the AA pairs were very high due to the tiny difference between one audio and another when the same prompt was used in the TTS, which only varied in the speech rate (85%, 100%, and 115%). The lowest AA pair's similarity was 0.9754, while the average was 0.99. As a result, the modules that rely on voice feature comparison should not be evaluated with synthesized audio.

4.5.1 Driver identification threshold definition

In the driver identification by features comparison module, the system instructs the driver to say its name. The recorded audio is compared to the recorded audios of driver's names, which characterizes an AA pair (same content and same speaker). That way, the minimum similarity of an AA pair in Table 20 was used to define the threshold of the driver identification module, which was previously set as 0.8, based on Graph 1. Since identification is made at system startup, the driver can confirm the recognized identity, so false positives are less concerning. Therefore, a threshold of 0.7 was chosen to reduce false negatives during the driver identification.

4.5.2 Speaker verification threshold definition

For the speaker verification, the recorded audio is compared to the driver embedding, which corresponds to a summary of the driver's voice, obtained by averaging the audio embeddings of the driver.

The threshold for the speaker verification module was defined by comparing the 272 calculated similarities (17 speakers $\times$ 16 pairs per speaker) to five predefined values, as presented

in Table 21. The similarities smaller than the threshold were counted, with the percentage that this number represents of the total pairs considered. We chose 0.6 due to the low count of similarities below it.

Table 21 – Number of similarities smaller than the threshold.

Threshold	Count	Percentage (%)
0.80	142	52.21
0.75	82	30.15
0.70	38	13.97
0.65	9	3.31
0.60	2	0.74

Source: Own authorship (2022).

4.5.3 Voice features similarity thresholds definition

The lowest similarity of an AA pair was 0.7536, which is smaller than the 0.8 previously defined, based on the histogram presented in Graph 1. It happened in another three cases. Because of this, threshold 1 for Algorithm 4 ($threshold1$) was reduced to 0.65.

The similarity of the AA pair corresponds to at least 88 % of the highest similarity in the 11 cases in which it is not the highest value. That way, a value of 0.85 was defined for $threshold2$ in Algorithm 4.

4.6 Spectrogram classifier threshold definition

The MobileNet spectrogram classifier model predictions made on the test dataset resulted in the confidence scores plotted in the histogram shown in Graph 7, which was used to define the best threshold for the vehicle actuation module ($threshold3$) to compare with the confidence score of each spectrogram classification.

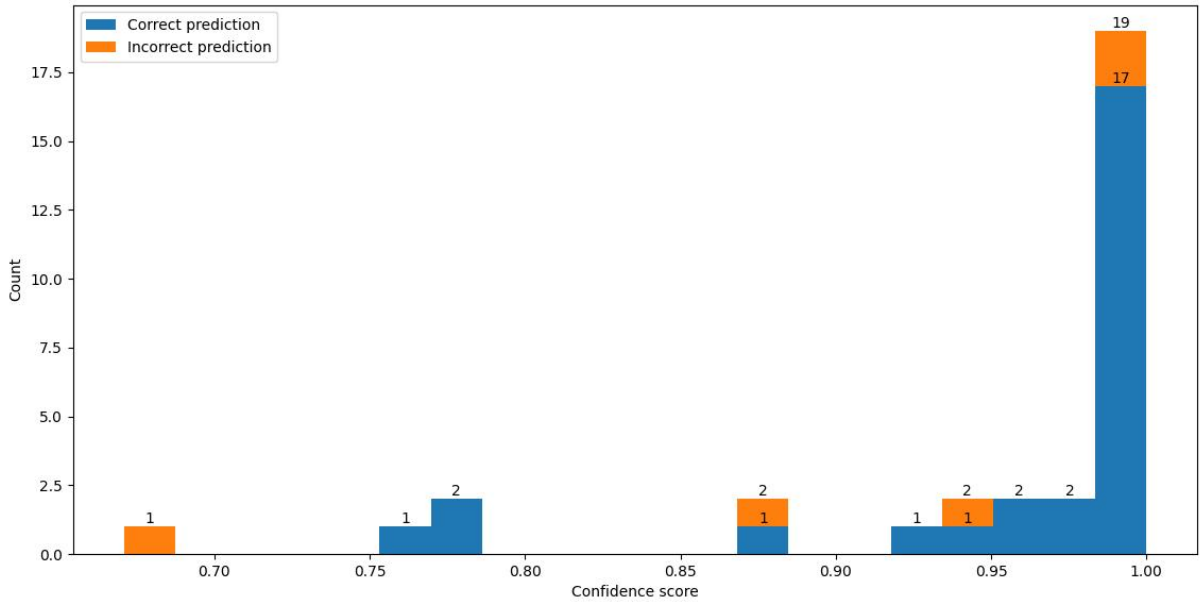
Five different thresholds were considered for counting the numbers of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). The results are presented in Table 22.

Table 22 – Average similarities between commands of the same speaker.

Threshold	TP	TN	FP	FN
0.95	21	3	2	6
0.9	23	2	3	4
0.85	24	1	4	3
0.8	24	1	4	3
0.75	27	1	4	0

Source: Own authorship (2022).

Graph 7 – Histogram of confidence scores of MobileNet predictions on test dataset.



Source: Own authorship (2022).

In the spectrogram classification, it is better to discard a correct classification with low confidence (FN) than to accept an incorrect one with high confidence (FP). In the first case, the other two command recognition subsystems may still recognize the correct command, but a wrong classification may interfere negatively in the decision-making of the vehicle actuation module.

That way, to minimize the number of FP, 0.95 was chosen to be used as $threshold3$ of Algorithm 4. Even with a high threshold, the classifier may realize incorrect predictions, besides discarding some correct predictions with confidence below the threshold. Consequently, we can assume that the spectrogram classification module may be unreliable to be used by the system, as it can trick the vehicle actuation module into actuating an erroneous vehicle function.

4.7 Components tests

Table 23 shows the checklist of the tests made to validate the developed software components.

Table 23 – Checklist of components tests.

Test Case	Req. ID	Expected Result	Status
CT01	R02.3	The introduction module explains how the button should be used to interact with the system	☑
CT02	R01.3	The audio recording module does not record when the button is not being pressed	☑
CT03	R01.3	The audio captured by the microphone is recorded to a *.wav* file	☑
CT04	R03.1	The audio processing module segments the audio into speech and non-speech sections	☑
CT05	R02.2	The driver report module speaks the input text	☑
CT06	R03	The audio features extraction module saves a *.npy* file with the input audio embeddings	☑
CT07	R04.1	The audio features extraction module saves a *.png* file with the input audio spectrogram	☑
CT08	R03.2	The features comparison module calculates the similarity of the audio features with a list of *.npy* files	☑
CT09	R03.4	The features storage module creates the needed folders	☑
CT10	R03.4	The features storage module saves each file to the corresponding folder	☑
CT11	R03.4	The select driver module finds the driver folder relative to the given ID	☑
CT12	R03.4	The driver data loading module loads all the *.npy* files from the driver folder to an array	☑
CT13	R03.3	The speaker verification module recognizes when the voice is not from the loaded driver	☑
CT14	R04.1	The STT module estimates the transcript of the audio	☑
CT15	R04.1	The STT module calculates the ASR evaluation metrics	☑
CT16	R04.1	The STT module outputs a single command	☑
CT17	R04.1	The voice features similarity results in an array with 4 similarity scores, one for each command	☑
CT18	R04.1	The voice features similarity module outputs a single command	☑
CT19	R04.1	The spectrogram classification module outputs a single command	☑
CT20	R04.3.2	The vehicle actuation module reads the simulated signals from the vehicle	☑
CT21	R04.3	The vehicle actuation module activate/deactivate the chosen vehicle system	☑
CT22	R03.4.1	The new features storage module appends the embeddings array to the chosen *.npy* file	☑
CT23	R03.4.1	The new features storage module saves the spectrogram image file	☑

Source: Own authorship (2022).

4.7.1 Audio recording module test

To test the audio recording module, 25 audios of each voice command were recorded by a single person, creating a new dataset with 100 audios that could also be used for testing other modules. The average recording time was 1.31 ($\pm$ 0.26) seconds, with the headlights command being the shorter (1.08 $\pm$ 0.11 seconds) and the right turn signal command being the longer (1.57 $\pm$ 0.16 seconds). The shortest audios were one command of headlights and one hazard warning command, both with 0.896 seconds.

4.7.2 Voice activity detection: audio processing module test

All audios from VVCD were used as input for the audio processing module of the VAD subsystem. The number of approved audios was counted to evaluate the module performance. The audio is accepted if at least 20% of its content is classified as speech.

The module classified 17 audios as non-speech, which corresponds to an error of 8.33%. It can be due to the recordings' fixed duration of four seconds, given that if the person speaks the command in less than 0.8 seconds, the speech corresponds to less than 20% of the audio. However, the duration of the recordings during the use of the system depends on how long the user holds down the button, which may reduce the error in system use. This assumption was proved by testing the 100 audios recorded with the audio recording module, which resulted in all being classified as speech audios.

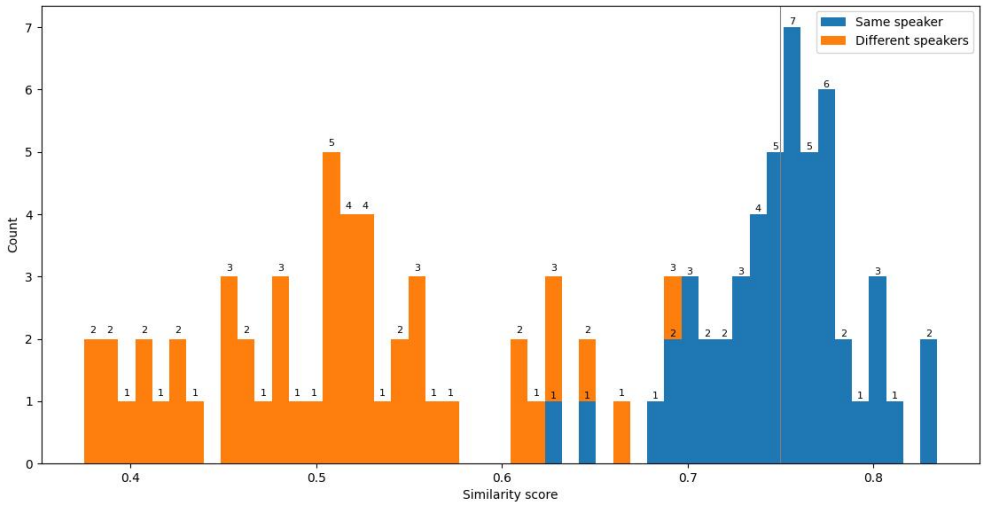
4.7.3 Driver identification: features comparison module test

The driver identification subsystem recognizes the driver through the features comparison module. It compares the input audio embeddings to all the embeddings of drivers' names recorded in the driver enrollment step. Since the participants of the dataset generation did not provide audio saying their names, the desired comparison is impossible. That way, since the driver identification is an example of a comparison of an AA pair (explained in Subsection 4.5.1), the headlights command audios were chosen to be used instead.

For each speaker in the VVCD, the summary of the headlights voice command was embedded in a *.npy* file by the audio features extraction module. The test consisted in selecting each speaker as the driver, then picking the three audios of the driver's voice command of headlights and three from the other speakers to compare to the *.npy* file. The histogram presented in Graph 8 shows the 102 calculated similarities (17 speakers $\times$ 6 comparisons).

With the threshold of 0.75 defined in Subsection 4.5.1, the driver identification module presented a total of 27 TP, 51 TN, 0 FP, and 24 FN, resulting in an accuracy of 77%. This histogram indicates that reducing the threshold to 0.7 improves the identification accuracy to 95%.

Graph 8 – Histogram of similarity scores of headlights command to simulate the driver identification.



Source: Own authorship (2022).

4.7.4 Speaker verification module test

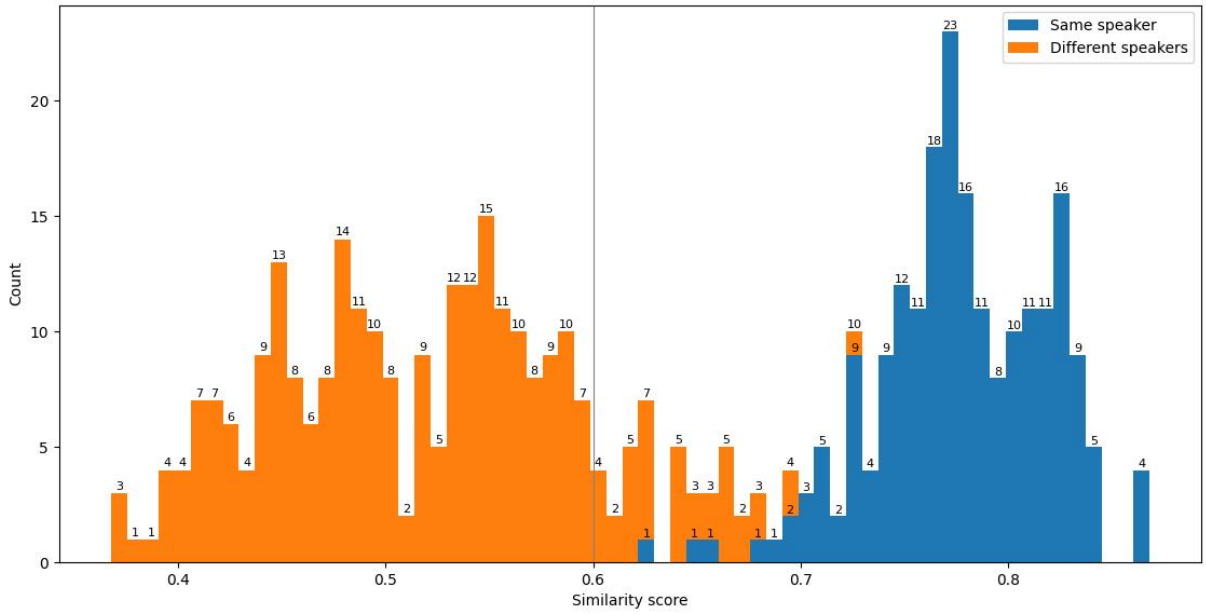
The speaker verification module is responsible for checking if the recorded voice is from the driver or not, not calling the command recognition modules in the last case. Because of this, it is better to try to recognize audio from other speakers (FP) than to discard one driver's command (FN), since this can even lead to an accident.

The audios of each of the 17 speakers were summarized, creating a *driver.npy* file for every "driver" to test the speaker verification module. This file was compared to all audios of the driver and one random audio from each of the other speakers, resulting in 28 comparisons by speaker, totaling 476 comparisons.

Most driver audios should be approved to minimize the False Rejection Rate (FRR). This way, the fewer false negatives there are, the better. The defined threshold of 0.6 was used to count TP (204), TN (234), FP (38), and FN (0) in the histogram of similarities calculated by comparing the speakers' embeddings, which is presented in Graph 9.

An accuracy of 0.92 was obtained by the speaker verification module, which means that it would correctly recognize when the voice is from the actual driver or not, more than 90% of the time.

Graph 9 – Histogram of similarity scores of the speaker verification module test.



Source: Own authorship (2022).

4.7.5 Speech-to-Text module test

The STT module tests used the 60 audios from the test subset of the VVCMD and the 100 audios recorded during the audio recording module test.

From the 160 analyzed audios, 147 transcripts were trustworthy since at least one word of the transcript matched a word of one command. From the reliable transcripts, 138 resulted in the correct command recognition, of which 115 were perfect transcriptions.

The module could not decide the command in eight cases, as the transcripts were like examples 2 and 3 from Table 8. One of these cases was from the left turn signal (*seta para esquerda*), and the obtained transcript was *seta para*, which resulted in the same word level measures for both the left and right turn signal commands. The other seven cases were similar, but the commands were neither left turn signal nor right turn signal, and the transcripts were *para* and *seta*, which induced the module to be indecisive between these two commands.

Five of the 22 incorrect commands were from confusion about which turn signal has to be activated. These cases were similar to example 1 of Table 8. All of them were the left turn signal command, and the module returned the right turn signal command because of the transcripts *seta para a* and *seta para a sala*.

4.7.6 Voice features similarity module test

The voice features similarity module calculates the similarity of the recorded voice command embedding to each command *.npy* file created in the driver enrollment. We selected two audios of each folder of VVCMD to generate the *.npy* files for this test. The selection criterion was that the audio was not present in the test set, as the test audios would be compared to the *.npy* files.

There were eight cases in which two audios of the same folder were in the test set, leaving just one to create the command embedding. That way, one audio of the test set was used to generate the command embedding and was not used to test the module.

From the 60 audios from the test set of VVCMD, 52 were used together with the 100 audios, recorded as explained in Subsection 4.7.1, as inputs of the voice features similarity module. The transcript obtained in the STT module test (Subsection 4.7.5) was used to calculate $score1$.

The module correctly identified the command in 134 of the 152 cases, but, as explained in Section 4.5, the synthesized audios are not recommended for testing modules that depend on voice similarity since they are almost the same audio, only varying the speed. That resulted in the module correctly recognizing the command in 100 % of the 30 synthesized audios, with an average similarity score of 0.9642 ($\pm$ 0.0156).

When considering the remaining 122 audios, the module correctly recognizes the command in 85% of the cases, with an average similarity score of 0.7911 ($\pm$ 0.0568). The maximum similarity was 0.9213, and the minimum was 0.6697.

4.7.7 Spectrogram classification module test

The remaining 122 audios (cited in Subsection 4.7.6) were used to test the spectrogram classification module, which succeeded in recognizing the command in 54 cases (44.26%). The average confidence was 0.9438 ($\pm$ 0.116). In 29 situations, the confidence score was below 0.95, the threshold defined in Section 4.6. As a result, even if the module correctly recognizes the command, it will not be considered in the decision-making of the vehicle actuation module. That occurred in six cases. From the 93 cases in which the actuation module would consider the spectrogram classification, 45 were wrong.

The classification module had unexpectedly low results, given the accuracy of 0.844

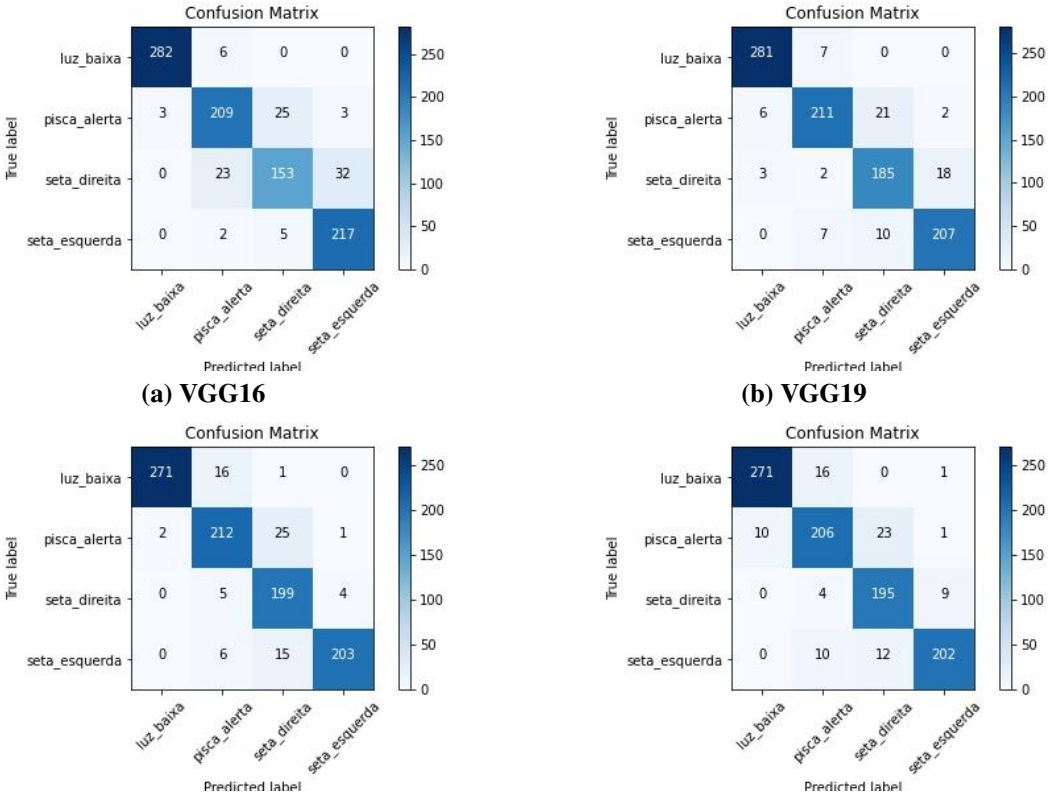
obtained by the spectrogram classifier model. The module's poor performance was associated with the reduced size of the dataset of spectrograms used for the model training. The dataset had only 204 images, divided into training (140 images), validation (32 images), and test (32 images) subsets, which means that the 0.844 accuracy obtained in the tests represents 27 correct predictions from a total of 32 images.

Thus, we decided to train four new models, based on the results of Table 18: VGG19 and MobileNet for presenting the best test accuracy (both with 0.844), VGG16 for the second best accuracy (0.781), and MobileNetV2 for being the smallest model, with an accuracy of 0.719.

For this training, each audio of the VVCMD was augmented using 15 sound effects combinations before the spectrogram generation. The effects used were: clipping, variations on the pitch, reverb, time dropouts, additive noise, and band-reject filter.

After the augmentations, the spectrogram database consisted of 6336 images and was divided into training (4432 images), validation (944 images), and test (960 images) subsets. The confusion matrices of the resulting models are presented in Graph 10.

Graph 10 – Confusion matrices of the new spectrogram classifier models.



(a) VGG16 **(b) VGG19**

(c) MobileNet **(d) MobileNetV2**

Source: Own authorship (2022).

The MobileNet stood out for having the best accuracy, with a value of 0.922, followed

by VGG19, with 0.921. MobileNetV2 had an accuracy of 0.91, which corresponds to an improvement of 26.56% when compared to the accuracy of the previous MobileNetV2 model. VGG16 was last, with an accuracy of 0.897. MobileNet improved by 9.24% over the previous model and was the chosen model again because of its results.

With the new MobileNet model, the spectrogram classification module recognized the command in 94 cases of 122 (77.05%), representing an improvement of 74.07% compared to the previous model. The average confidence was 0.9603 ($\pm$ 0.101).

In 103 cases, it got a confidence score higher than 0.95, but it predicted the wrong command in 19 of these cases. It predicted the correct command in 10 cases where the score was below the threshold, which happened in 19 situations, meaning that these predictions were not used for the final command estimation. Considering only the remaining 103 cases in which the spectrogram classification would be used, the module was correct in 84 predictions, corresponding to 81.55% of the cases.

4.7.8 Vehicle actuation module test

The results of the tests of the three command recognition modules (speech-to-text, voice features similarity, and spectrogram classification) were used for testing the vehicle actuation module. The module was tested two times: first, considering only the STT and voice features similarity modules, and then considering all the three command recognition modules.

It actuated correctly in 120 of the 122 evaluated cases, corresponding to 98.36% of correct recognition of voice commands. No command was actuated in the two other cases, which is better than actuating in the wrong vehicle system. In both cases, the command from the STT module was considered unreliable, which makes the module return a Null command, as explained in Algorithm 1.

In one of these cases, the voice features comparison command was correct, with a similarity score of 0.91. In contrast, the command estimated by the STT module had a similarity of 0.74, corresponding to 81% of the highest score. The spectrogram classification module also got the correct command, with a confidence of 1.

In the other case where the module did not act on any system in the vehicle, the spectrogram classification module got the correct command, with a confidence of 0.99.

The results of the actuation module are the same, whether the spectrogram classifier is being used or not. That way, it was turned off since its use impacts the system performance,

increasing the processing time. The average increase was 0.187 ($\pm$ 0.028) seconds, which is very relevant, considering that the average of the summed processing times of the three command recognition modules plus the actuation module decision-making algorithm is 0.234 ($\pm$ 0.030) seconds. As a result, the spectrogram classifier processing time corresponds to almost 80% of the time spent by the command recognition step (after the speaker verification).

4.8 System tests (V1)

As explained in Subsection 3.1.4.1, the hardware used for the first version of the system was only a notebook with built-in speakers and microphone. There was no need for integration with other hardware, as the vehicle actuation step was simulated in the first version of the system. That way, the architecture was already validated, and the integration tests were unnecessary.

The VA software was built following the definitions and requirements presented in Subsection 3.1.3. After that, the system was tested according to Frame 2. The results of these tests are presented in Table 24.

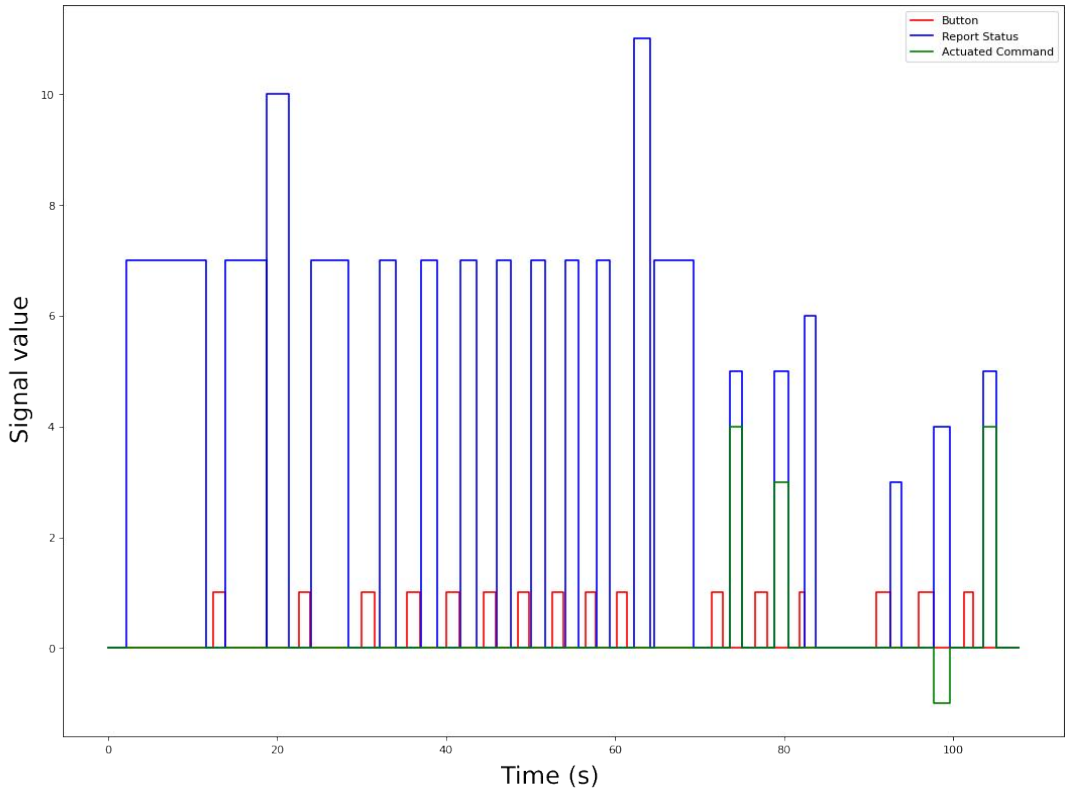
Table 24 – Checklist of system (V1) tests.

Test Case	Req. ID	Expected Result	Status
ST01	R02	The system indicates when it is activated	☑
ST02	R01.3	The system records audio only when required	☑
ST03	R03.1	The system verify if someone spoke	☑
ST04	R03.1.1	The system reports when no speech was detected	☑
ST05	R03.2	The system identify the driver through its voice	☑
ST06	R03.2.1	The system reports when the driver was not identified	☑
ST07	R03.2.2	The system reports when it identifies the driver	☑
ST08	R03.3	The system verify if the driver spoke	☑
ST09	R03.3.1	The system reports when the voice fails the verification	☑
ST10	R04.1	The system recognize the voice commands	☑
ST11	R04.1.1	The system reports if a voice command was not recognized	☑
ST12	R04.2	The system does not actuate if the command was not said by the driver	☑
ST13	R04.3	The system actuate if the command was recognized	☑
ST14	R04.3.1	The system reports when a command was received	☑

Source: Own authorship (2022).

Some signals (variables) were stored in a log file to check if the expected results of the system tests were achieved: the Button, the Actuated Command, and the Report Status, which was explained in Subsection 3.1.2.1. Graph 11 shows the log of the first run of the system.

Graph 11 – First test of the system (V1).



Source: Own authorship (2022).

The system has eleven different report statuses:

1. The driver was correctly identified;
2. The voice was not identified;
3. The voice failed verification;
4. The voice was verified, but the command was not recognized;
5. The voice was verified, and the command was recognized;
6. No voice activity was detected;
7. The system is giving instructions to the user;
8. The driver chose option 1 when asked by the identification subsystem;
9. The driver chose option 2 when asked by the identification subsystem;
10. New driver enrollment started;

11. New driver enrollment completed.

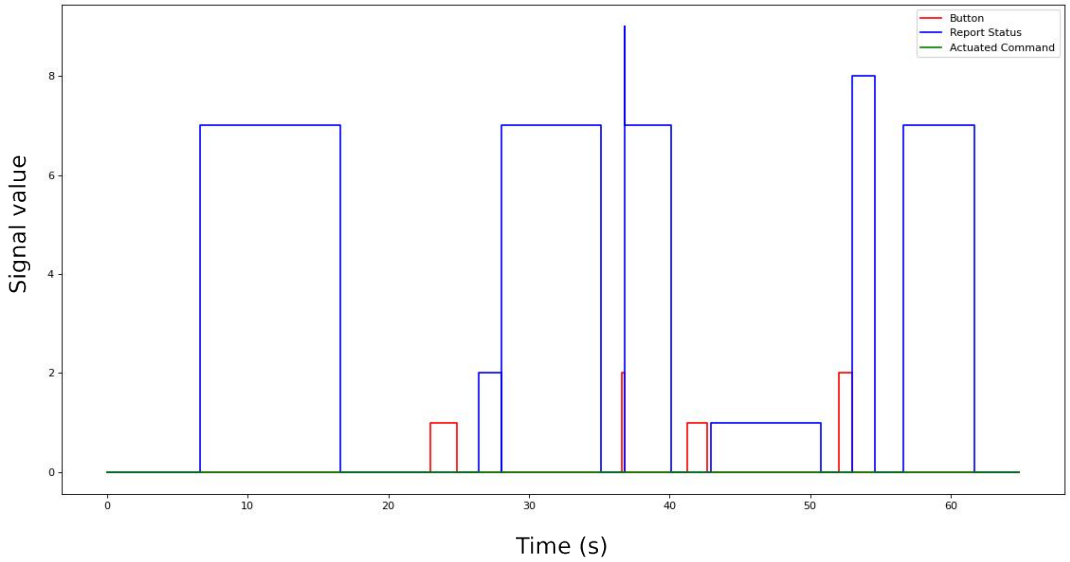
The tests were performed by a driver and a passenger, to simulate the execution in a vehicle. The system began by instructing the user how to interact with it (status 7). The instructions oriented the driver to say his name for identification and to press the button each time before speaking. It also presented the sound that indicates it is ready for user interaction.

After the first time that the driver pushed the button to speak his name, the system detected that there were no registered drivers, and the enrollment process began (status 10). The following nine button presses were for the driver to repeat his name, followed by two audios of each command. The system gave instructions to the user on how to proceed to the following enrollment step after each recording. By the end of the enrollment (status 11), the system explained that it was ready to receive voice commands.

In the following two button presses, the driver spoke two different voice commands (headlights and hazard warning), which were correctly recognized by the system (status 5), activating both functions (actuated command 4 and 3, respectively). After that, the button was pressed for a short time, and the driver did not speak (status 6). Then, the driver held the button, but the passenger who spoke, and the system detected that the voice was not from the driver (status 3). Afterward, the driver held the button and said something other than a command, which resulted in the system reporting that it did not recognize the command (status 4 and actuated command equals -1). Finally, the driver gave another headlights command, which the system correctly recognized (status 5), deactivating this function (actuated command 4).

Graph 12 shows the log of the second system test, which was performed to evaluate the identification process. The system started the same way as before, instructing the user how to interact with it (status 7). In this test, the passenger was speaking during the first button press. Therefore, the system has not recognized the voice as one of the registered drivers, failing identification (status 2). When this happens, the system asks the user if it should start a new enrollment or retry the identification. The driver chose the second option (status 9) and said his name on the subsequent button pressing. This time, the system identified the voice (status 1) and asked for confirmation of the recognized ID, which was confirmed by the driver (status 8). The test ended with the system informing that it was ready to receive commands from the driver (status 7). The Button signal (represented in red) assumes the value of 2 when the user is answering one system prompt the value 1 the other times it is pressed.

Graph 12 – Test of the identification process in the first version of the system.



Source: Own authorship (2022).

The third test was to check the processing time of the system. We used two computers to perform this test at the same time. Computer 1 is the same one used for the previous tests, whose specifications were presented in Frame 4. The specifications of computer 2 are presented in Table 25.

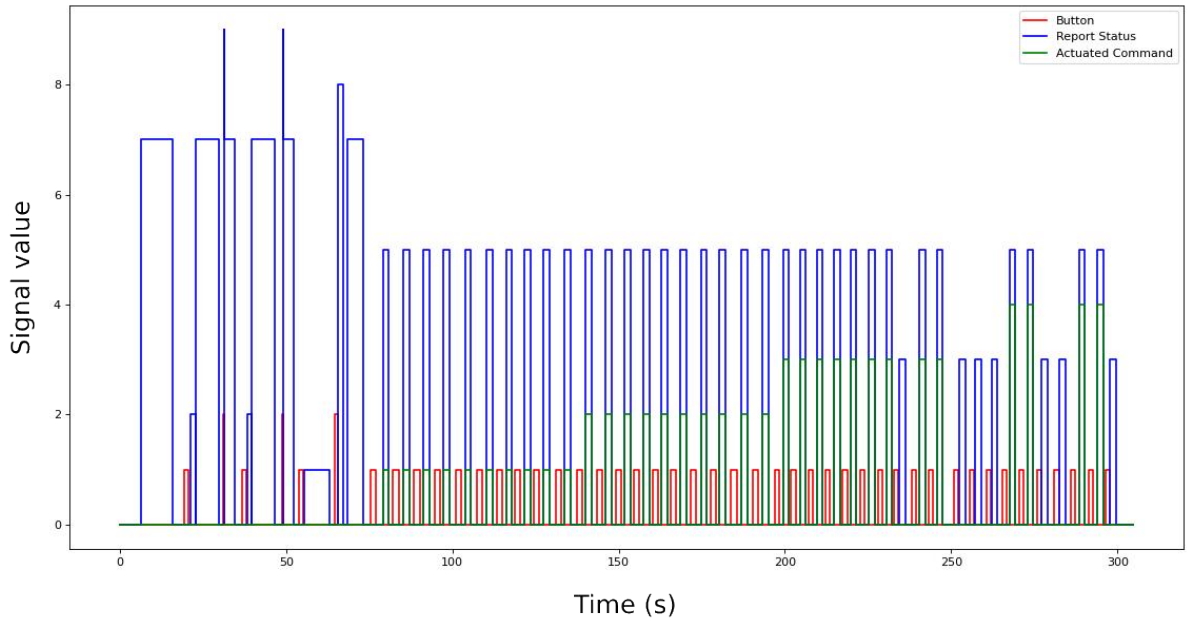
Table 25 – Specifications of computer 2.

Component	Specification
Microprocessor	Intel® Core™ i7-10750H (2.6GHz, up to 5GHz) 12MB Cache
Memory, standard	16GB, DDR4 2933MHz (1 x 16 GB)
Video Graphics	NVIDIA GeForce RTX 2070 Max-Q 8GB
OS	Windows 10
Storage	512 GB SSD (M2 NVMe)

Source: Own authorship (2022).

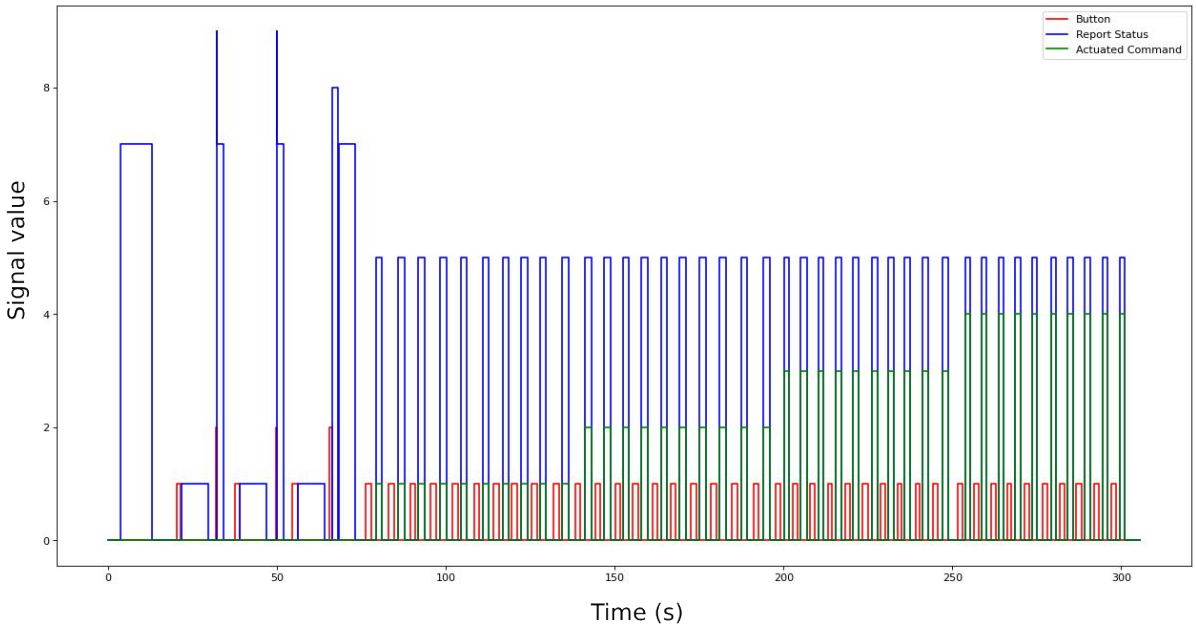
The logs of the third system test executed on computers 1 and 2 are presented in Graph 13 and Graph 14, respectively. To perform the test, the driver (already enrolled in the first test) started the system and said each command ten times after being identified by the system. Both computers were executing the same software version, and the systems were launched together. The system loaded faster in the second computer with 3.5 seconds against 6.3 seconds from computer 1.

Graph 13 – Processing time test executed on computer 1.



Source: Own authorship (2022).

Graph 14 – Processing time test executed on computer 2.



Source: Own authorship (2022).

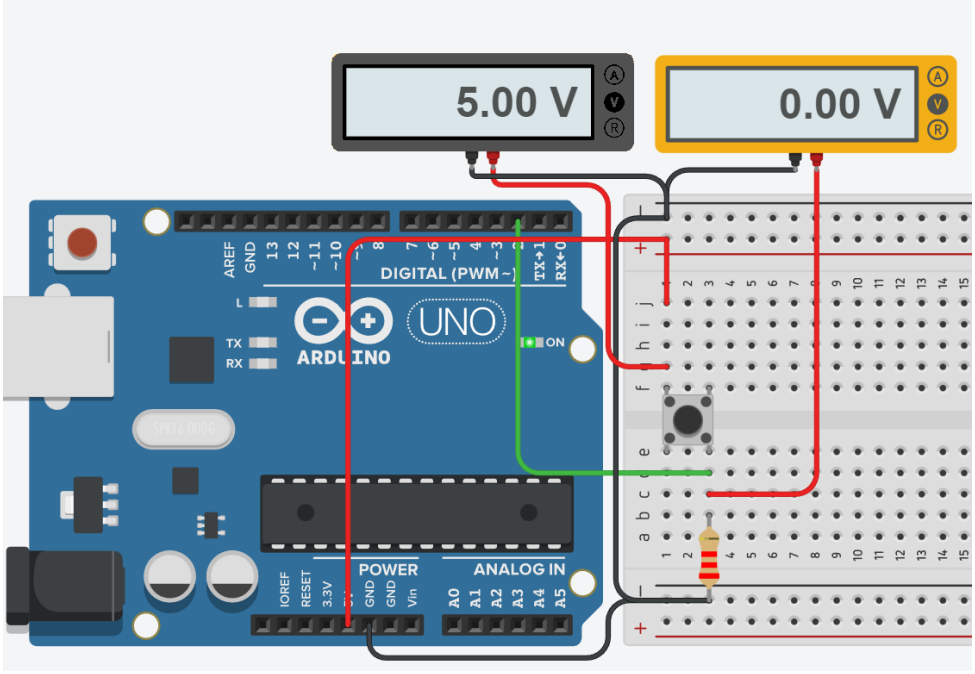
The system failed to identify the voice two times in computer 1, with similarities lower than the identification threshold of 0.7 (0.68 and 0.69, respectively). It also failed to verify the voice once in the hazard warning command and six times in the headlights commands. The average similarity in the cases rejected by the speaker verification process was 0.58, while the verification threshold was 0.6.

There were no failures in the system running on computer 2, which makes it likely that the problems presented by computer 1 were caused by the microphone or other hardware, as both computers were running the same software at the same time, and receiving the same speeches. The system correctly recognized the commands whenever the voice was approved by the speaker verification process, which happened in 82.5% of the commands on computer 1 and 100% of the commands on computer 2. In terms of processing time, the command recognition took an average of 1.17 seconds on computer 1 and 1.01 seconds on computer 2.

4.9 Integration tests (V2)

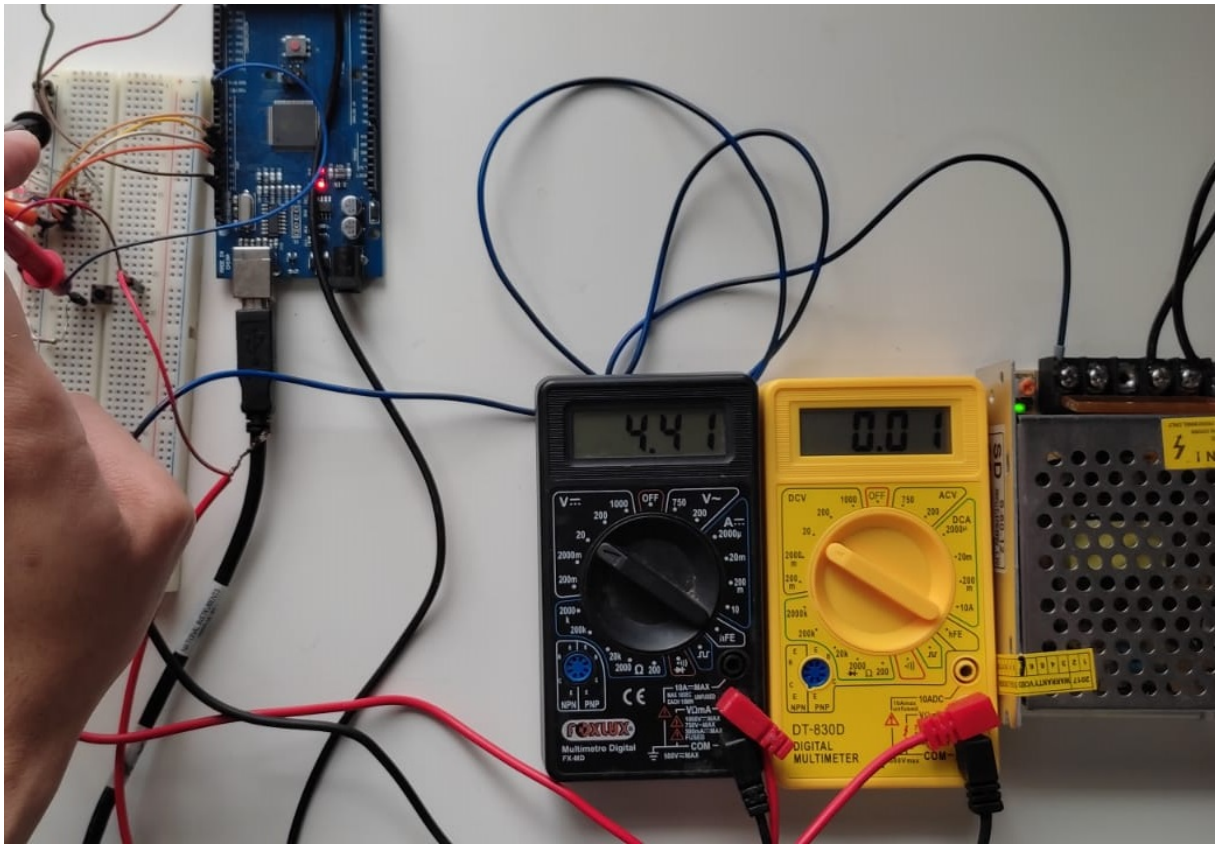
For the integration tests, we used two multimeters in the voltmeter function to measure the voltage level of the tested connections. To perform the integration tests IT03, IT04, and IT05, the multimeters were connected to the protoboard, as shown in Figure 30. The tests are represented by Photographs 1 and 2.

Figure 30 – Multimeters connection for the button testing.



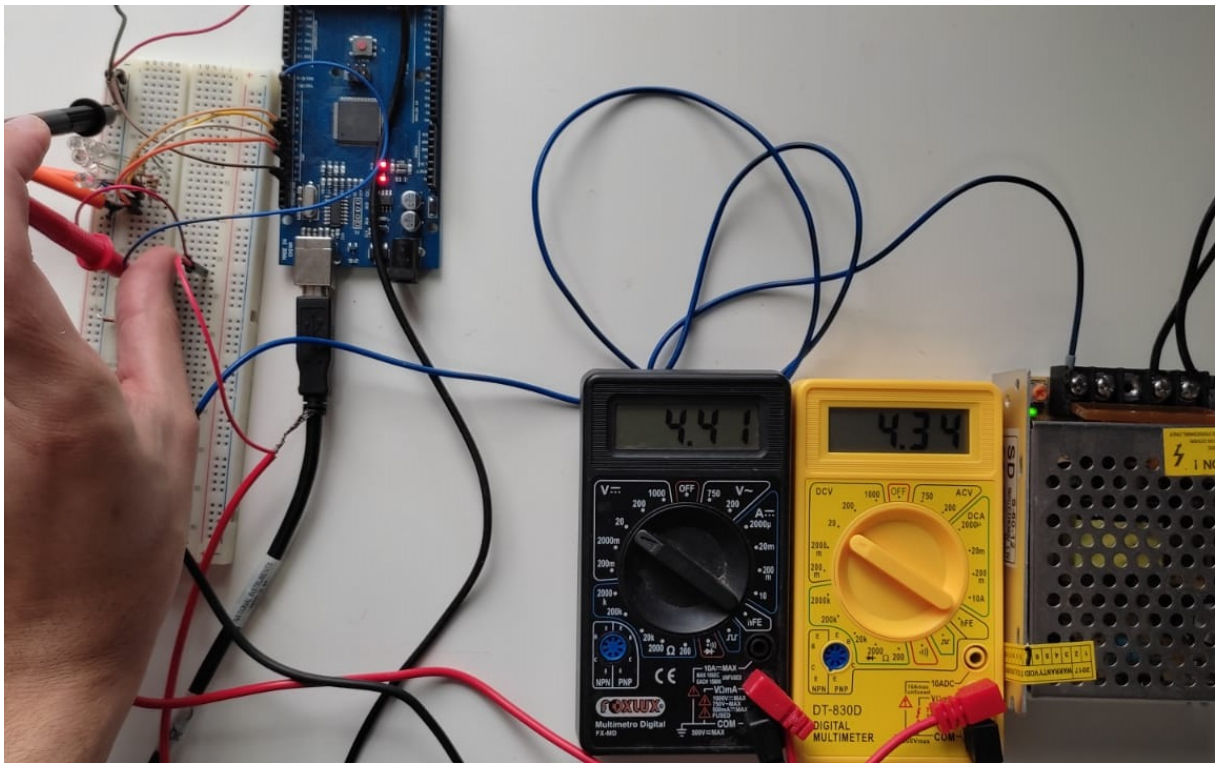
Source: Own authorship (2022).

Photograph 1 – Hardware integration test with the button not pressed.



Source: Own authorship (2022).

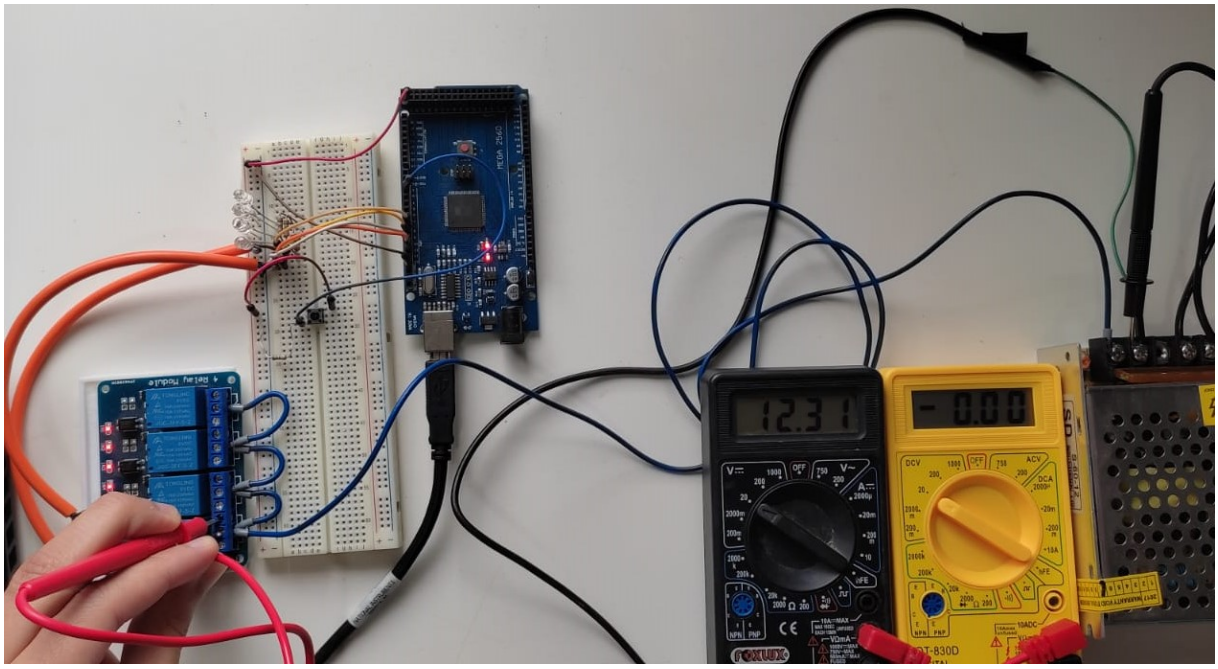
Photograph 2 – Hardware integration test with button pressed.



Source: Own authorship (2022).

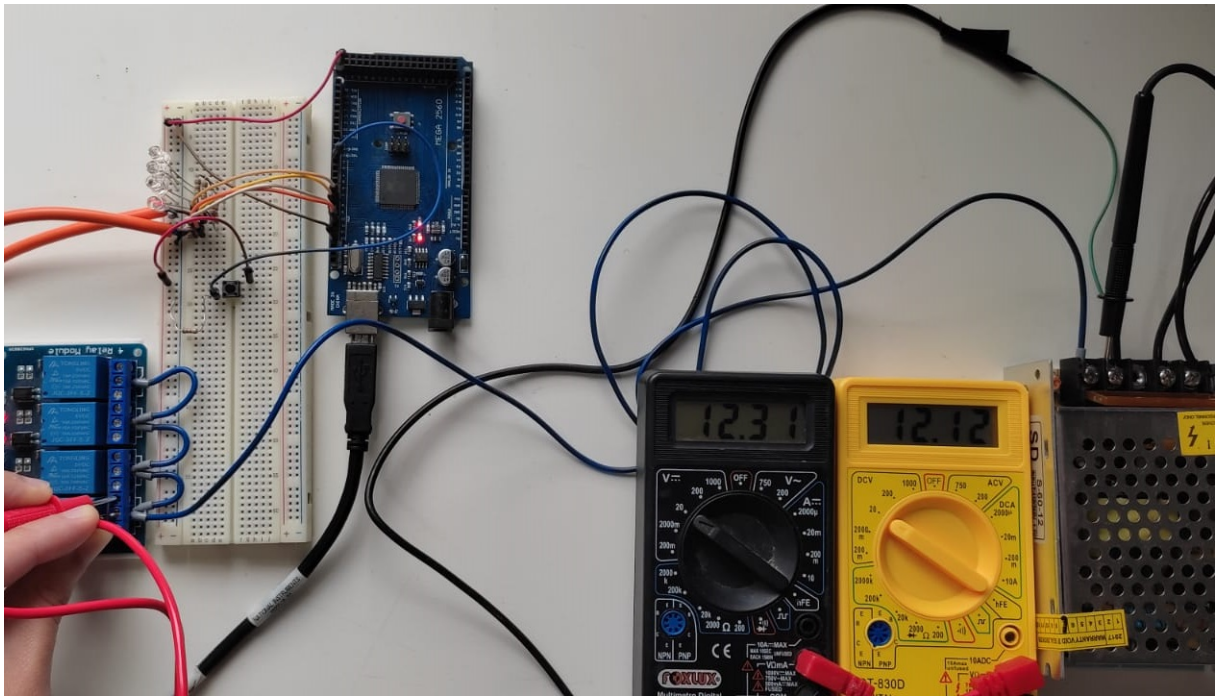
The integration tests IT07, IT08, and IT09, are represented by Photographs 3 and 4. The black multimeter was connected to the relay's common pin, and the yellow multimeter was connected to the normally open pin.

Photograph 3 – Hardware integration test with relay off.



Source: Own authorship (2022).

Photograph 4 – Hardware integration test with relay on.



Source: Own authorship (2022).

The results of the integration tests for the second version of the system are presented in Frame 8. The system passed all tests.

Frame 8 – Checklist of integration tests.

Test Case	Req. ID	Expected Result	Status
IT01	R01	The microphone picks up audio for the computer	☑
IT02	R02.2	The computer outputs audio through the speakers	☑
IT03	R01.2	The push button receives 3.5V or more from the microcontroller	☑
IT04	R01.2	The push button outputs 1V or less to the microcontroller when not pressed	☑
IT05	R01.2	The push button outputs 3.5V or more to the microcontroller when pressed	☑
IT06	R04.3.2	The computer reads a 16 bits signal by serial port from the microcontroller actuation system	☑
IT07	R04.3	The actuation system relays receives 12V or more from the car battery	☑
IT08	R04.2	The actuation system relays output 1V or less when not activated	☑
IT09	R04.3	The actuation system relays output 12V or more when activated	☑
IT10	R04.3	The microcontroller receives a 16 bits signal by serial port from the computer actuation system	☑

Source: Own authorship (2022).

For the serial communication between the microcontroller and the computer running the system, we used 2 bytes to send data from one device to the other, which attends the expected results from tests IT06 and IT10.

4.10 System tests (V2)

After testing the hardware integration, the system was tested again to ensure that the second version worked well. The results of the tests of the second version of the system are presented in Frame 9.

Frame 9 – Checklist of system (V2) tests.

Test Case	Req. ID	Expected Result	Status
ST01	R02	The system indicates when it is activated	☑
ST02	R01.3	The system records audio only when required	☑
ST03	R03.1	The system verify if someone spoke	☑
ST04	R03.1.1	The system reports when no speech was detected	☑
ST05	R03.2	The system identify the driver through its voice	☑
ST06	R03.2.1	The system reports when the driver was not identified	☑
ST07	R03.2.2	The system reports when it identifies the driver	☑
ST08	R03.3	The system verify if the driver spoke	☑
ST09	R03.3.1	The system reports when the voice fails the verification	☑
ST10	R04.1	The system recognize the voice commands	☑
ST11	R04.1.1	The system reports if a voice command was not recognized	☑
ST12	R04.2	The system does not actuate if the command was not said by the driver	☑
ST13	R04.3	The system actuate if the command was recognized	☑
ST14	R04.3.1	The system reports when a command was received	☑

Source: Own authorship (2022).

This time, the system was tested inside a Renault Captur 2018. The ambient noise was measured with a decibel meter Instrutherm DEC-415 (Photograph 5(a)). Before measuring, the decibel meter was calibrated using an acoustic calibrator Akrom KR94 (Photograph 5(b)).

Photograph 5 – Decibel meter and acoustic calibrator.

(a) Decibel meter Instrutherm DEC-415 **(b) Acoustic calibrator Akrom KR94**

Source: Own authorship (2022).

The calibration process is shown in Photograph 6. First, the decibel meter was turned on, and its microphone was inserted into the acoustic calibrator. Then the calibrator was set to 94 dB (decibels), and the decibel meter calibration trimpot was adjusted to read 94 dB. The exact process was repeated for 114 dB. The decibel meter was considered calibrated when both measurements were equal to the value set on the calibrator.

Photograph 6 – Decibel meter calibration.

(a) 94 dB calibration **(b) 114 dB calibration**

Source: Own authorship (2022).

The first test used white noise in the vehicle auto speakers. The noise amplitude was controlled, and the sound level was measured with the decibel meter. Each command was pronounced five times in all tested sound levels, totaling 20 voice commands for each level. The results are presented in Table 26.

Table 26 – System test with controlled white noise.

Sound level (dB)	Correct	Failed verification	Failed recognition
47.1	20	0	0
63.6	20	0	0
78.2	19	0	1
91.5	19	1	0
102.5	17	3	0

Source: Own authorship (2022).

The system failed to recognize one voice command at 78.2 dB, with no command being actuated. In higher sound levels, the system failed to verify the driver's voice once at 91.5 dB and three times at 102.5 dB. However, these two noise levels are considered too high and do

not represent real driving scenarios. Besides, according to annex 1 of the Brazilian regulatory standard NR 15, these noise levels are not recommended for humans to keep listening for more than 3 hours, considering 91.5 dB (BRASIL, 2022). When considering the 102.5 dB level, the time reduces to 45 minutes.

After these tests, the system was tested in six real driving scenarios:

(A) Vehicle systems ON;

(B) Vehicle ON, stopped and without people talking;

(C) Vehicle ON, stopped and with passengers talking;

(D) Vehicle moving, without people talking;

(E) Vehicle moving, with passengers talking, shown in Photograph 7;

(F) Vehicle moving, with music playing.

Photograph 7 – Test of the system in a real driving scenario.

Source: Own authorship (2022).

In all test cases, the vehicle's front windows were open. The vehicle traveled on a highway for test cases D, E, and F. Table 27 shows how the system performed in these six test cases and the average sound level measured with the decibel meter.

Table 27 – System test in real case scenarios.

Test case	Sound level (dB)	Correct	Failed verification	Failed recognition
A	40.1	20	0	0
B	47.4	20	0	0
C	64.9	20	0	0
D	73.8	20	0	0
E	81.2	19	0	1
F	84.1	19	0	1

Source: Own authorship (2022).

The system worked perfectly up to an average of 73.8 dB, presented in the situation with the vehicle traveling a highway, with no internal conversations, which resembles the average sound level presented by vehicles under similar conditions in the study of Louw (2017). The system starts to fail when the noise level approaches 80 dB, just as it did in the tests with white noise. In 81.2 dB, the system failed to recognize one voice command. The same happened in the last test case, where the average sound level was 84.1 dB. The two commands that were not recognized resulted in no commands being actuated in the vehicle, which is better than actuating the wrong command.

Considering the results from Tables 26 and 27, the system correctly actuated in 96.8% of the voice commands. In addition, the processing time for recognizing commands was calculated, with an average of 0.9414 ($\pm$ 0.1721) seconds. The average recording time was 1.3352 ($\pm$ 0.2239) seconds.

5 CONCLUSIONS AND FUTURE WORKS

This Master's thesis addresses the design and development of a voice assistant to recognize voice commands in Brazilian Portuguese and control functions of a vehicle dashboard, mainly to help drivers with reduced mobility. The design process was based on the v-model of software development since it is widely used in the automotive industry.

In this work, we collected data from 18 audio datasets in Brazilian Portuguese and contributed by creating a new dataset with voice commands for operating dashboard functions. We also developed a tool called Audio Dataset Creator to assist in the creation of voice datasets. It automatically validates the recorded audio by comparing the recording prompt to the obtained transcript. Besides that, the main contribution of this work is the developed system, which enables some vehicle functions to be controlled by voice commands.

Before the development of the voice assistant, three versions of the system were described in the design stage: two prototypes and the MVP, which consists of the system running on a vehicle. This work focused on developing the first two versions to validate the designed system on a PoC.

There is sufficient evidence to support the claim of hypothesis H1, which states that speech similarity can be used to estimate voice commands, as shown in the results in Subsection 4.7.6, in which this technique has successfully recognized 85% of the voice commands.

Despite the improvement in the spectrogram classifier module achieved with the new MobileNet model, the results are not sufficient to validate hypothesis H2 since the improvement in the command recognition could not be observed. However, the results of the classifier model tests indicated that it could be used for voice command recognition.

The results of the system prototypes were promising, attending to all established requirements, reaching an accuracy of 100% of recognized commands during the tests of the first version of the system tests, with more than 98% during the component tests. The second version of the system correctly recognized 96.8% of the voice commands in a vehicle in real driving scenarios, even with high noises. Thus, the main objective of the work could be achieved, proving that a voice assistant can be used to control the dashboard functions of an adapted vehicle for PRM, helping the driver.

For future work, the third version of the system must be developed and tested in a vehicle in real driving scenarios, with the system controlling the vehicle dashboard. The use of

filters and other digital signal processing techniques might be investigated as well. Also, speaker localization techniques can be tested in the vehicle, as it may enhance the speaker verification process since it was responsible for most of the failures in the tests of the second version of the system prototype. Besides that, acceptance testing of the final product must be performed. Furthermore, the actuation module decision-making algorithm could be substituted by an ANN model.

REFERENCES

ALENCAR, V. F. S.; ALCAIM, A. Lsf and lpc - derived features for large vocabulary distributed continuous speech recognition in brazilian portuguese. *In*: **2008 42nd Asilomar Conference on Signals, Systems and Computers**. *[S.l.: s.n.]*, 2008. p. 1237–1241.

ALVAREZ, Ignacio; HAMMOND, Aqueasha Martin; DUNBAR, Jerone; TAIBER, Joachim; WILSON, Dale-Marie; GILBERT, Juan. Voice interfaced vehicle user help. *In*: **Proceedings of 2nd International Conference on Automotive User Interfaces and Interactive Vehicular Applications, AutomotiveUI 2010**. *[S.l.: s.n.]*, 2010. p. 42–49.

APPEN. **Appen**. 2022. Available at: https://appen.com.

ASSOCIATION, ELRA European Language Resources. **Globalphone Portuguese (Brazilian) – elra catalogue**. 2018. Available at: https://catalog.elra.info/en-us/repository/browse/ELRA-S0201/.

BAI, Zhongxin; ZHANG, Xiao-Lei. **Speaker Recognition Based on Deep Learning: An Overview**. arXiv, 2020. Available at: https://arxiv.org/abs/2012.00931.

BRASIL. Nr 15 - atividades e operações insalubres. **Ministério do Trabalho e Previdência**, p. 2–3, Apr. 2022.

BRAUN, Michael; MAINZ, Anja; CHADOWITZ, Ronee; PFLEGING, Bastian; ALT, Florian. At your service: Designing voice assistant personalities to improve automotive user interfaces. *In*: **2019 CHI Conference**. *[S.l.: s.n.]*, 2019. p. 1–11.

BRUNETE, Alberto; GAMBAO, Ernesto; HERNANDO, Miguel; CEDAZO, Raquel. Smart assistive architecture for the integration of iot devices, robotic systems, and multimodal interfaces in healthcare environments. **Sensors**, v. 21, n. 6, 2021. ISSN 1424-8220. Available at: https://www.mdpi.com/1424-8220/21/6/2212.

CHATFIELD, Ken; SIMONYAN, Karen; VEDALDI, Andrea; ZISSERMAN, Andrew. **Return of the Devil in the Details: Delving Deep into Convolutional Nets**. arXiv, 2014. Available at: https://arxiv.org/abs/1405.3531.

CHEN, Yu hsin; LOPEZ-MORENO, Ignacio; SAINATH, Tara N.; VISONTAI, Mirkó; ALVAREZ, Raziel; PARADA, Carolina. Locally-connected and convolutional neural networks for small footprint speaker recognition. *In*: **Interspeech 2015**. ISCA, 2015. Available at: https://doi.org/10.21437/interspeech.2015-297.

CHUNG, Joon Son; ZISSERMAN, Andrew. Out of time: Automated lip sync in the wild. *In*: **ACCV Workshops**. *[S.l.: s.n.]*, 2016.

DAHLBäCK, N.; JöNSSON, A.; AHRENBERG, L. Wizard of oz studies — why and how. **Knowledge-Based Systems**, v. 6, n. 4, p. 258–266, 1993. ISSN 0950-7051. Special Issue: Intelligent User Interfaces. Available at: https://www.sciencedirect.com/science/article/pii/095070519390017N.

DANIEL, POVEY; ARNAB, GHOSHAL; GILLES, BOULIANNE; LUKáš, BURGET; ONDřEJ, GLEMBEK; K., GOEL Nagendra; MIRKO, HANNEMANN; PETR, MOTLÍČEK; YANMIN, QIAN; PETR, SCHWARZ; JAN, SILOVSKÝ; GEORG, STEMMER; KAREL, VESELÝ. The kaldi speech recognition toolkit. *In*: . *[S.l.: s.n.]*, 2011.

FENDJI, Jean Louis; TALA, Diane; YENKE, Blaise; ATEMKENG, Marcellin. Automatic speech recognition and limited vocabulary: A survey. **ArXiv**, 03 2022.

FIOCRUZ. **Painel de Indicadores**. 2019. Available at: https://www.pns.icict.fiocruz.br/painel-de-indicadores-mobile-desktop/.

Grupo FalaBrasil. 2020. Available at: https://ufpafalabrasil.gitlab.io/.

HANNUN, Awni; CASE, Carl; CASPER, Jared; CATANZARO, Bryan; DIAMOS, Greg; ELSEN, Erich; PRENGER, Ryan; SATHEESH, Sanjeev; SENGUPTA, Shubho; COATES, Adam; NG, Andrew Y. Deep speech: Scaling up end-to-end speech recognition. **arXiv**, 2014. Available at: https://arxiv.org/abs/1412.5567.

HEAFIELD, Kenneth. KenLM: Faster and smaller language model queries. *In*: **Proceedings of the Sixth Workshop on Statistical Machine Translation**. Edinburgh, Scotland: Association for Computational Linguistics, 2011. p. 187–197. Available at: https://aclanthology.org/W11-2123.

HEIGOLD, Georg; MORENO, Ignacio; BENGIO, Samy; SHAZEER, Noam. End-to-end text-dependent speaker verification. *In*: **2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. *[S.l.: s.n.]*, 2016. p. 5115–5119.

HOY, Matthew B. Alexa, siri, cortana, and more: An introduction to voice assistants. **Medical Reference Services Quarterly**, Routledge, v. 37, n. 1, p. 81–88, 2018. PMID: 29327988.

IBGE. **Perfil dos municípios brasileiros : 2020 / IBGE**. IBGE, 2021. ISBN 978-65-87201-87-0. Available at: https://biblioteca.ibge.gov.br/index.php/biblioteca-catalogo?view=detalhes&id=2101871.

JEMINE, Corentin. **Real-Time Voice Cloning**. 2019. 14 p. Master's Thesis (Master's Thesis) — Faculté des Sciences appliquées, 2019.

JIA, Ye; ZHANG, Yu; WEISS, Ron J.; WANG, Quan; SHEN, Jonathan; REN, Fei; CHEN, Zhifeng; NGUYEN, Patrick; PANG, Ruoming; MORENO, Ignacio Lopez; WU, Yonghui. Transfer learning from speaker verification to multispeaker text-to-speech synthesis. **arXiv**, 2018. Available at: https://arxiv.org/abs/1806.04558.

JUNIOR, Arnaldo Candido; CASANOVA, Edresson; SOARES, Anderson; OLIVEIRA, Frederico Santos de; OLIVEIRA, Lucas; JUNIOR, Ricardo Corso Fernandes; SILVA, Daniel Peixoto Pinto da; FAYET, Fernando Gorgulho; CARLOTTO, Bruno Baldissera; GRIS, Lucas Rafael Stefanel; ALUíSIO, Sandra Maria. Coraa: a large corpus of spontaneous and prepared speech manually validated for speech recognition in brazilian portuguese. arXiv, 2021.

Keras Team. **Keras Documentation: Keras applications**. 2022. Available at: https://keras.io/api/applications/.

KINGMA, Diederik P.; BA, Jimmy. Adam: A method for stochastic optimization. **arXiv**, 2014. Available at: https://arxiv.org/abs/1412.6980.

KOEHN, Philipp. Europarl: A parallel corpus for statistical machine translation. *In*: **Proceedings of Machine Translation Summit X: Papers**. Phuket, Thailand: *[s.n.]*, 2005. p. 79–86. Available at: https://aclanthology.org/2005.mtsummit-papers.11.

LANDER, T.; COLE, Ronald A.; OSHIKA, B. T.; NOEL, M. The OGI 22 language telephone speech corpus. *In*: **Proc. 4th European Conference on Speech Communication and Technology (Eurospeech 1995)**. *[S.l.: s.n.]*, 1995. p. 817–820.

LEI, Xin; SENIOR, Andrew W.; GRUENSTEIN, Alexander; SORENSEN, Jeffrey Scott. Accurate and compact large vocabulary speech recognition on mobile devices. *In*: **INTERSPEECH**. *[S.l.: s.n.]*, 2013.

LILLICRAP, Timothy P; SANTORO, Adam. Backpropagation through time and the brain. **Current Opinion in Neurobiology**, v. 55, p. 82–89, 2019. ISSN 0959-4388. Machine Learning, Big Data, and Neuroscience. Available at: https://www.sciencedirect.com/science/article/pii/S0959438818302009.

LIMA, Thales Aguiar de; COSTA-ABREU, Márjory da. A survey on automatic speech recognition systems for portuguese language and its variations. **Computer Speech & Language**, v. 62, p. 63, 2020. ISSN 0885-2308. Available at: https://www.sciencedirect.com/science/article/pii/S0885230819302992.

LIN, Shih-Chieh; HSU, Chang-Hong; TALAMONTI, Walter; ZHANG, Yunqi; ONEY, Steve; MARS, Jason; TANG, Lingjia. Adasa: A conversational in-vehicle digital assistant for advanced driver assistance features. *In*: **Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology**. New York, NY, USA: Association for

Computing Machinery, 2018. (UIST '18), p. 531–542. ISBN 9781450359481. Available at: https://doi.org/10.1145/3242587.3242593.

LISBOA, Yasmin. Deficientes físicos relatam as dificuldades de conseguir transporte por aplicativo. **Diário da Região**, Jan 2021. Available at: https://www.diariodaregiao.com.br/cidades/deficientes-fisicos-relatam-as-dificuldades-de-conseguir-transporte-por-aplicativo-1.26041.

LOUW, Nicol. **How quiet is your car's cabin? Three vehicles tested**. 2017. Available at: https://www.carmag.co.za/technical/technical/how-quiet-is-your-cars-cabin-three-vehicles-tested/.

LUKIC, Yanick; VOGT, Carlo; DURR, Oliver; STADELMANN, Thilo. Speaker identification and clustering using convolutional neural networks. *In*: **2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)**. IEEE, 2016. Available at: https://doi.org/10.1109/mlsp.2016.7738816.

MAHAJAN, Kirti; LARGE, David R.; BURNETT, Gary; VELAGA, Nagendra R. Exploring the effectiveness of a digital voice assistant to maintain driver alertness in partially automated vehicles. **Traffic Injury Prevention**, Taylor & Francis, v. 22, n. 5, p. 378–383, 2021. PMID: 33881365. Available at: https://doi.org/10.1080/15389588.2021.1904138.

MASINA, Fabio; ORSO, Valeria; PLUCHINO, Patrik; DAINESE, Giulia; VOLPATO, Stefania; NELINI, Cristian; MAPELLI, Daniela; SPAGNOLLI, Anna; GAMBERINI, Luciano. Investigating the accessibility of voice assistants with impaired users: Mixed methods study. **Journal of medical Internet research**, JMIR Publications, v. 22, n. 9, p. e18431–e18431, Sep 2020. ISSN 1438-8871.

MATARNEH, Rami; MAKSYMOVA, Svitlana; LYASHENKO, Vyacheslav; BELOVA, Nataliya. Speech recognition systems: A comparative review. **IOSR Journal of Computer Engineering**, v. 19, p. 71–79, 10 2017.

MCCULLOCH, Warren S.; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity. **The Bulletin of Mathematical Biophysics**, Springer Science and Business Media LLC, v. 5, n. 4, p. 115–133, Dec. 1943. Available at: https://doi.org/10.1007/bf02478259.

MCFEE, Brian; LOSTANLEN, Vincent; METSAI, Alexandros; MCVICAR, Matt; BALKE, Stefan; THOMé, Carl; RAFFEL, Colin; ZALKOW, Frank; MALEK, Ayoub; LEE, Kyungyun; AL. et. **Librosa/librosa: 0.8.0**. 2020. Available at: https://doi.org/10.5281/zenodo.3955228.

Mobilize Brasil. **CALÇADAS DO BRASIL 2019: Uma avaliação da caminhabilidade nas cidades brasileiras**. *[S.l.]*, 2019. Available at: https://www.mobilize.org.br/campanhas/calcadas-do-brasil-2019/.

MORAIS, Reuben. **External scorer scripts**. 2020. Available at: https://deepspeech.readthedocs.io/en/v0.9.3/Scorer.html.

MORAIS, Reuben; DAVIS, Kelly. **Deepspeech model**. 2020. Available at: https://deepspeech.readthedocs.io/en/master/DeepSpeech.html.

MORAIS, Reuben; KAMP, Tilman; THIELE, Olaf; SAGAR, Karan; MEYER, Josh; ORBIK, Jędrzej Beniamin; JONES, Dewi Bryn; EBERHARDT, Christian; WAŁEJKO Łukasz. **Training your own model**. 2020. Available at: https://deepspeech.readthedocs.io/en/v0.9.3/TRAINING.html#augmentation.

MORGAN, John; ACKERLIND, Sheila; PACKER, Sterling. **West Point Brazilian Portuguese Speech**. Linguistic Data Consortium, 2008. Available at: https://catalog.ldc.upenn.edu/LDC2008S04.

MORRIS, Andrew; MAIER, Viktoria; GREEN, Phil. From wer and ril to mer and wil: improved evaluation measures for connected speech recognition. *In*: **INTERSPEECH**. *[S.l.: s.n.]*, 2004.

MOZILLA. **Mozilla Common Voice**. 2022. Available at: https://commonvoice.mozilla.org/en/datasets.

NAGRANI, Arsha; CHUNG, Joon Son; ZISSERMAN, Andrew. VoxCeleb: A Large-Scale Speaker Identification Dataset. *In*: **Proc. Interspeech 2017**. *[S.l.: s.n.]*, 2017. p. 2616–2620.

OLIVEIRA, Luiza Maria Borges; Secretaria de Direitos Humanos da Presidência da República (SDH/PR); Secretaria Nacional de Promoção dos Direitos da Pessoa com Deficiência (SNPD); Coordenação-Geral do Sistema de Informações sobre a Pessoa com Deficiência. Cartilha do censo 2010 – pessoas com deficiência. *In*: ____. **SDH-PR/SNPD**. Brasilia, Brasil: SDH-PR/SNPD, 2012. Available at: https://inclusao.enap.gov.br/wp-content/uploads/2018/05/cartilha-censo-2010-pessoas-com-deficienciareduzido-original-eleitoral.pdf.

PANAYOTOV, Vassil; CHEN, Guoguo; POVEY, Daniel; KHUDANPUR, Sanjeev. Librispeech: An asr corpus based on public domain audio books. *In*: **2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. *[S.l.: s.n.]*, 2015. p. 5206–5210.

PRADHAN, Alisha; MEHTA, Kanika; FINDLATER, Leah. "accessibility came by accident": Use of voice-controlled intelligent personal assistants by people with disabilities. *In*: ____. **Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems**. New York, NY, USA: Association for Computing Machinery, 2018. p. 1–13. ISBN 9781450356206. Available at: https://doi.org/10.1145/3173574.3174033.

PRATAP, Vineel; XU, Qiantong; SRIRAM, Anuroop; SYNNAEVE, Gabriel; COLLOBERT, Ronan. MLS: A large-scale multilingual dataset for speech research. **ArXiv**, abs/2012.03411, 2020.

QUINTANILHA, Igor; NETTO, Sergio; BISCAINHO, Luiz. An open-source end-to-end ASR system for brazilian portuguese using DNNs built from newly assembled corpora. **Journal of Communication and Information Systems**, Sociedad Brasileira de Telecomunicacoes, v. 35, n. 1, p. 230–242, 2020. Available at: https://doi.org/10.14209/jcis.2020.25.

QUINTANILHA, Igor Macedo; BISCAINHO, Luiz Wagner Pereira; NETTO, Sergio Lima. Towards an end-to-end speech recognizer for portuguese using deep neural networks. *In*: **XXXV SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES E PROCESSAMENTO DE SINAIS - SBrT2017**. São Pedro (SP), Brazil: *[s.n.]*, 2017. p. 408–412. Available at: http://www02.smt.ufrj.br/~sergioln/papers/BC31.pdf.

ROOK, Paul. Controlling software projects. **Software Engineering Journal**, v. 1, p. 7–16(9), January 1986. ISSN 0268-6961.

SAINATH, Tara; PARADA, Carolina. Convolutional neural networks for small-footprint keyword spotting. *In*: **Interspeech**. *[S.l.: s.n.]*, 2015.

SALESKY, Elizabeth; WIESNER, Matthew; BREMERMAN, Jacob; CATTONI, Roldano; NEGRI, Matteo; TURCHI, Marco; OARD, Douglas W.; POST, Matt. Multilingual tedx corpus for speech recognition and translation. *In*: **Proceedings of Interspeech**. *[S.l.: s.n.]*, 2021.

SCHMIDT, Maria; MINKER, Wolfgang; WERNER, Steffen. How users react to proactive voice assistant behavior while driving. *In*: **LREC**. *[S.l.: s.n.]*, 2020.

SCHRAMM, Mauricio C.; FREITAS, Luis Felipe R.; ZANUZ, Adriano; BARONE, Dante. Cslu: Spoltech brazilian portuguese version 1.0. **Linguistic Data Consortium**, 2006. Available at: https://catalog.ldc.upenn.edu/LDC2006S16.

SHAH, Ayush; GUSIKHIN, Anastacia. Integration of voice assistant and smartdevicelink to control vehicle ambient environment. *In*: INSTICC. **Proceedings of the 6th International Conference on Vehicle Technology and Intelligent Transport Systems - VEHITS,**. *[S.l.]*: SciTePress, 2020. p. 522–527. ISBN 978-989-758-419-0. ISSN 2184-495X.

SIMONYAN, Karen; ZISSERMAN, Andrew. Very deep convolutional networks for large-scale image recognition. **arXiv 1409.1556**, 09 2014.

SNYDER, David; GARCIA-ROMERO, Daniel; POVEY, Daniel; KHUDANPUR, Sanjeev. Deep Neural Network Embeddings for Text-Independent Speaker Verification. *In*: **Proc. Interspeech 2017**. *[S.l.: s.n.]*, 2017. p. 999–1003.

SNYDER, David; GARCIA-ROMERO, Daniel; SELL, Gregory; POVEY, Daniel; KHUDANPUR, Sanjeev. X-vectors: Robust dnn embeddings for speaker recognition. *In*: **2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. *[S.l.: s.n.]*, 2018. p. 5329–5333.

SNYDER, David; GHAHREMANI, Pegah; POVEY, Daniel; GARCIA-ROMERO, Daniel; CARMIEL, Yishay; KHUDANPUR, Sanjeev. Deep neural network-based speaker embeddings for end-to-end speaker verification. *In*: **2016 IEEE Spoken Language Technology Workshop (SLT)**. *[S.l.: s.n.]*, 2016. p. 165–170.

TATOEBA. **Tatoeba: Collection of sentences and translations**. 2022. Available at: https://tatoeba.org/.

TERZOPOULOS, George; SATRATZEMI, Maya. Voice assistants and smart speakers in everyday life and in education. **Informatics in Education**, Vilnius University Institute of Data Science and Digital Technologies, v. 19, n. 3, p. 473–490, 2020. ISSN 1648-5831.

TIRUMALA, Sreenivas Sremath; SHAHAMIRI, Seyed Reza; GARHWAL, Abhimanyu Singh; WANG, Ruili. Speaker identification features extraction methods: A systematic review. **Expert Systems with Applications**, Elsevier BV, v. 90, p. 250–271, Dec. 2017. Available at: https://doi.org/10.1016/j.eswa.2017.08.015.

VARIANI, Ehsan; LEI, Xin; MCDERMOTT, Erik; MORENO, Ignacio Lopez; GONZALEZ-DOMINGUEZ, Javier. Deep neural networks for small footprint text-dependent speaker verification. *In*: **2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. *[S.l.: s.n.]*, 2014. p. 4052–4056.

VITERBI, A. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. **IEEE Transactions on Information Theory**, v. 13, n. 2, p. 260–269, 1967.

VOXFORGE. **VoxForge**. 2022. Available at: http://www.voxforge.org/.

WAN, Li; WANG, Quan; PAPIR, Alan; MORENO, Ignacio Lopez. Generalized end-to-end loss for speaker verification. **arXiv**, 2017. Available at: https://arxiv.org/abs/1710.10467.

WANG, Wei; HU, Yiyang; ZOU, Ting; LIU, Hongmei; WANG, Jin; WANG, Xin. A new image classification approach via improved mobilenet models with local receptive field expansion in shallow layers. **Computational Intelligence and Neuroscience**, Hindawi, v. 2020, p. 8817849, Aug 2020. ISSN 1687-5265. Available at: https://doi.org/10.1155/2020/8817849.

WARDEN, Pete. Speech commands: A dataset for limited-vocabulary speech recognition. **ArXiv**, 2018.

WYSE, L. Audio spectrogram representations for processing with convolutional neural networks. **arXiv**, 2017. Available at: https://arxiv.org/abs/1706.09559.

APPENDIX A - MANUAL DE USUÁRIO

APPENDIX A – MANUAL DE USUÁRIO

O sistema permite controlar funções do painel de instrumentos do veículo por meio de comandos de voz. Com o assistente de voz é possível acionar e desacionar a iluminação externa (luz baixa), as setas e também o pisca alerta do veículo, sem deslocar as mãos.

A.1 Funcionalidades do sistema

O sistema é capaz de reconhecer comandos de voz em Português Brasileiro, sendo eles:

- Seta para direita;
- Seta para esquerda;
- Luz baixa;
- Pisca alerta.

Ao pronunciar um comando, o sistema verifica o estado atual da função no veículo, acionando o sinal caso esteja desligado e desacionando caso esteja ligado.

O sistema é equipado com um botão, que é utilizado para iniciar a gravação de voz, não gravando áudios nos demais casos.

O sistema possui funcionalidades de reconhecimento de voz, sendo capaz de identificar o motorista por sua voz e também verificar se o comando foi falado pelo motorista, evitando que passageiros acionem alguma função do veículo indevidamente.

Além disso, o sistema é capaz de se comunicar de forma sonora para interagir com o usuário nas seguintes situações:

- Instruir o usuário a respeito da utilização do sistema;
- Solicitar interação do motorista;
- Indicar quando está pronto para receber comandos;
- Emitir relatórios de voz a respeito do funcionamento do sistema.

Os relatórios de voz são divididos em 11 classificações diferentes, como mostrado na tabela 10. Cada tipo de relatório corresponde a uma fala do assistente de voz.

Frame 10 – Tipos de relatórios de voz e falas do assistente de voz.

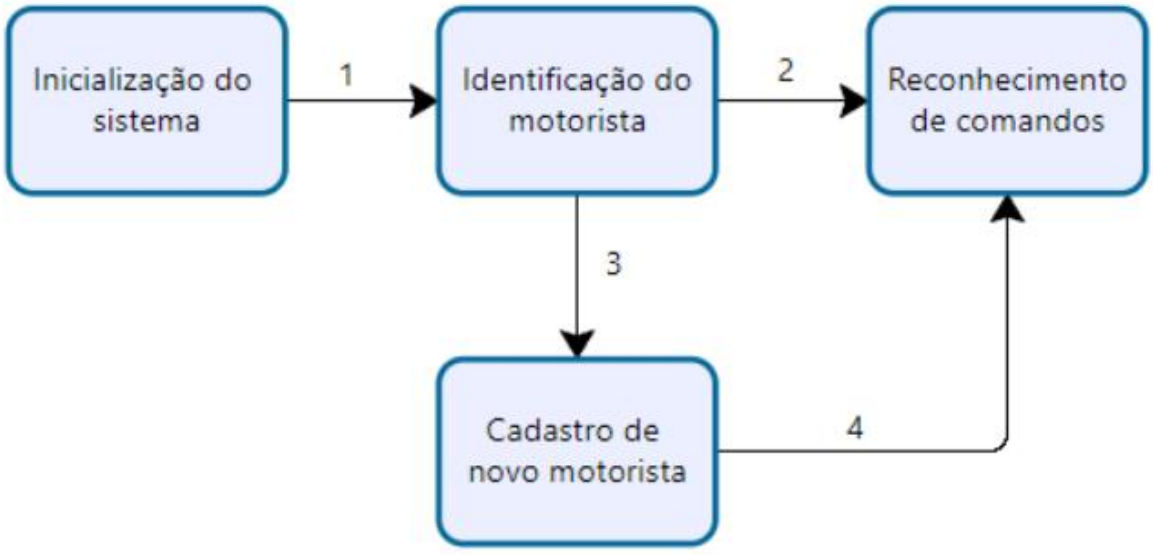
Tipo de relatório de voz	Fala do assistente de voz
O motorista foi corretamente identificado	Motorista identificado. Número de ID {número de ID}. Está correto? Segure o botão por 1 segundo para confirmar, ou pressione o botão para fazer a identificação novamente.
O motorista confirmou a identidade	Motorista confirmado.
O motorista optou por realizar a identificação novamente	Por favor, fale seu nome novamente para que eu possa identificá-lo.
A voz não foi identificada	Não reconheci seu nome. Deseja iniciar cadastro de novo motorista? Segure o botão por 1 segundo para confirmar, ou pressione o botão para fazer a identificação novamente.
O motorista optou por iniciar um novo cadastro	Você optou por iniciar cadastro de novo motorista. Seu número de identificação (ID) será {número de ID}. Por favor, repita seu nome para que eu aprenda sua voz.
A voz foi reprovada pelo processo de verificação	Voz não pertence ao motorista.
O comando não foi reconhecido	Comando não reconhecido.
O comando foi reconhecido	*
Não foi detectada atividade de voz	Não detectei voz.
Cadastro de novo motorista iniciado	Cadastro iniciado. ID: {número de ID}
Cadastro de novo motorista concluído	Cadastro concluído com sucesso.

* Nesse caso o assistente irá falar qual comando foi reconhecido e também se está acionando ou desacionando. Por exemplo: Caso o motorista fale "luz baixa" e a mesma esteja desligada, o assistente de voz irá falar "acionando luz baixa" e irá ligar a luz baixa. Porém, caso o sinal esteja ligado, o assistente falará "desacionando luz baixa" e irá desligar a luz baixa.

A.2 Etapas do funcionamento

O sistema possui 4 etapas principais, que são apresentadas na figura 31.

Figure 31 – Etapas do funcionamento do assistente de voz.



Na inicialização, o sistema é carregado. Após o carregamento, o assistente de voz instrui o usuário a respeito da utilização do sistema, orientando que o mesmo segure o botão para falar e diga seu nome para ser identificado.

Na identificação do motorista, o sistema tenta reconhecer o motorista por sua voz, atribuindo um número de identificação (ID) e solicitando confirmação do motorista. Caso a identificação falhe, o sistema pergunta se o usuário deseja iniciar o cadastro de um novo motorista ou realizar a identificação novamente.

O cadastro de novo motorista é iniciado caso o motorista não seja identificado pelo sistema e opte por iniciar um novo cadastro. Além disso, o cadastro é iniciado automaticamente pelo sistema caso ainda não existam motoristas cadastrados. No cadastro, o sistema orienta que o motorista diga novamente seu nome e, em seguida, fale duas vezes cada comando. Este processo dura aproximadamente 1 minuto.

A etapa de reconhecimento de comandos é executada após o motorista ser identificado por sua voz ou após o cadastro de um novo motorista ser concluído. O sistema indica que está pronto para receber comandos por meio de um aviso sonoro. Após cada comando de voz gravado, o sistema verifica se a voz é realmente do motorista, reconhecido na etapa de identificação ou de cadastro. Caso a voz seja do motorista, o comando é processado e a função do veículo é acionada ou desacionada de acordo com o comando reconhecido e do estado atual da função do veículo.

As transições entre etapas, indicadas na figura 31, significam, respectivamente:

1. O sistema finalizou as instruções ao usuário;
2. O motorista confirmou a identidade reconhecida pelo sistema;
3. O motorista optou por iniciar o cadastro de novo motorista, ou o sistema reconheceu que não possui motoristas cadastrados;
4. O cadastro de novo motorista foi concluído.

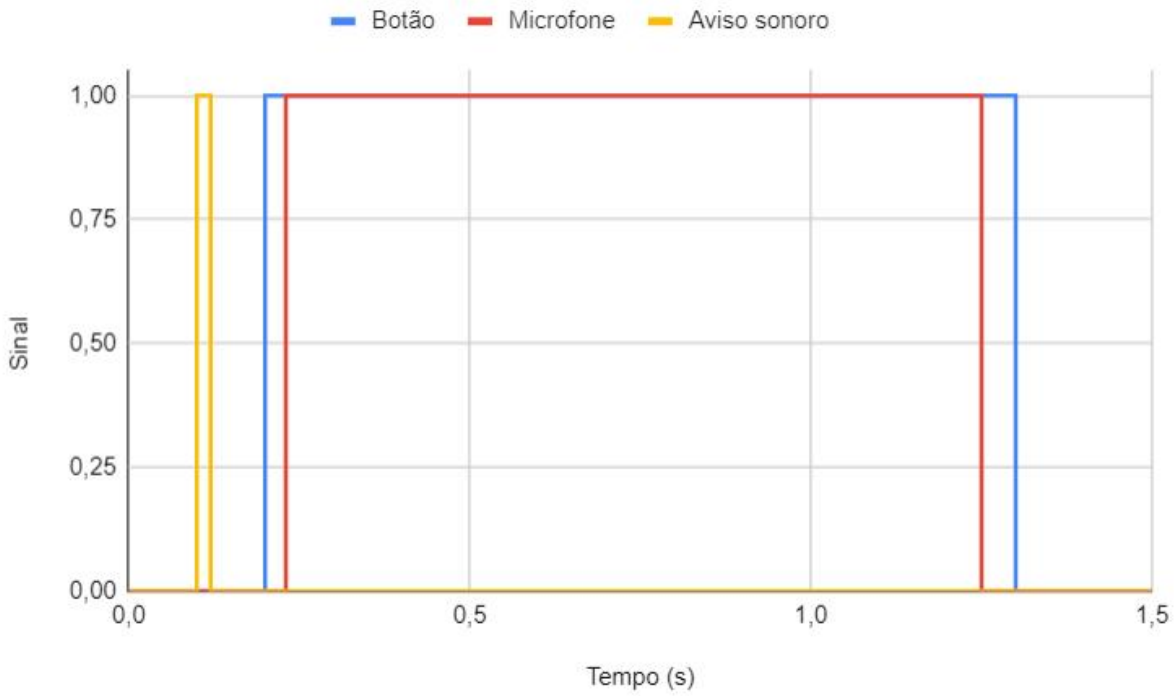
A.3 Como utilizar o sistema?

A identificação do motorista e o cadastro de novo motorista devem ser realizadas com o veículo parado.

A interação com o sistema é feita por meio de um botão e um microfone. O usuário deve aguardar o aviso sonoro que indica que o sistema está pronto, para pressionar o botão.

Para gravar áudios de comandos ou do nome (para a identificação), o usuário deve segurar o botão por tempo suficiente para todo o áudio, como mostra o gráfico 15.

Graph 15 – Forma adequada de interagir com o sistema.



Além disso, o botão também é utilizado para responder solicitações do sistema no processo de identificação do motorista.

Caso o sistema identifique a voz, será perguntado se o usuário deseja confirmar a identidade reconhecida ou realizar uma nova identificação. Para confirmar a identidade, o usuário deve segurar o botão por 1 segundo. Para realizar uma nova identificação, o usuário deve pressionar o botão.

Caso o sistema não identifique a voz, será perguntado se o usuário deseja iniciar o cadastro de um novo motorista ou tentar a identificação novamente. Para iniciar o cadastro de um novo motorista, o usuário deve segurar o botão por 1 segundo. Para realizar uma nova identificação, o usuário deve pressionar o botão.

APPENDIX B - USER MANUAL

APPENDIX B – USER MANUAL

The system allows controlling the vehicle's dashboard functions using voice commands. With the voice assistant it is possible to switch on and off the exterior lighting (headlights), the turn signals, and also the hazard warning signal of the vehicle, without displacing the hands.

B.1 System features

The system is able to recognize voice commands in Brazilian Portuguese:

- *Seta para direita* (Right turn signal);
- *Seta para esquerda* (Left turn signal);
- *Luz baixa* (Headlights);
- *Pisca alerta* (Hazard warning).

When a command is spoken, the system checks the current state of the function in the vehicle, activating the signal if it is off and deactivating it if it is on.

The system is equipped with a button, which is used to start voice recording, not recording audio in other cases.

The system has speaker recognition features, being able to identify the driver by his or her voice and also to verify if the command was spoken by the driver, preventing passengers from improperly activating some function of the vehicle.

In addition, the system is able to communicate audibly to interact with the user in the following situations:

- Instruct the user on how to use the system;
- Request interaction from the driver;
- Indicate when it is ready to receive commands;
- Give voice reports about the system's operation.

The voice reports are divided into 11 different classifications, as shown in the table 11. Each report type corresponds to one speech of the voice assistant.

Frame 11 – Types of voice reports and voice assistant speech.

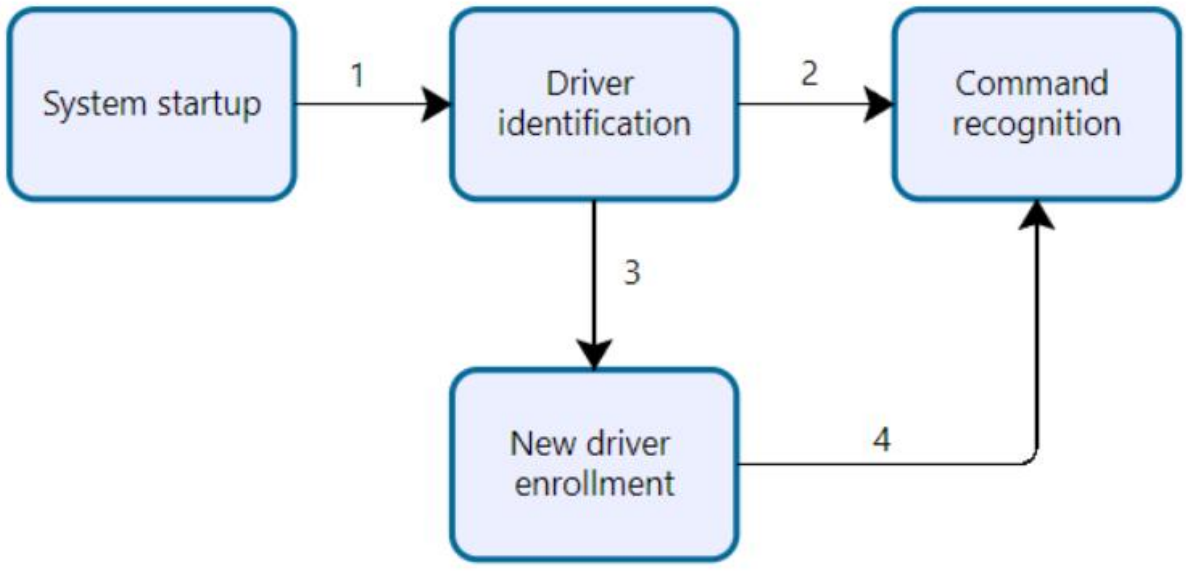
Voice report type	Voice assistant speech
The driver was correctly identified	*Motorista identificado. Número de ID* {ID number}. *Está correto? Segure o botão por 1 segundo para confirmar, ou pressione o botão para fazer a identificação novamente.*
The driver confirmed the identity	*Motorista confirmado.*
The driver chose to repeat the identification	*Por favor, fale seu nome novamente para que eu possa identificá-lo.*
The voice was not identified	*Não reconheci seu nome. Deseja iniciar cadastro de novo motorista? Segure o botão por 1 segundo para confirmar, ou pressione o botão para fazer a identificação novamente.*
The driver chose to start a new enrollment	*Você optou por iniciar cadastro de novo motorista. Seu número de identificação (ID) será* {ID number}. *Por favor, repita seu nome para que eu aprenda sua voz.*
The voice failed the verification process	*Voz não pertence ao motorista.*
The command was not recognized	*Comando não reconhecido.*
The command was recognized	*
No voice activity detected	*Não detectei voz.*
New driver enrollment started	*Cadastro iniciado. ID:* {ID number}
New driver enrollment completed	*Cadastro concluído com sucesso.*

* In this case the assistant will tell which command was recognized and also whether it is activating or deactivating. For example: If the driver says "*luz baixa*" and the headlights are off, the voice assistant will say "*acionando luz baixa*" and will turn on the headlights. However, if the signal is on, the assistant will say "*desacionando luz baixa*" and will turn off the headlights.

B.2 Operating steps

The system has 4 main steps, which are presented in the figure 32.

Figure 32 – Steps of the voice assistant operation.



At startup, the system is loaded. After loading, the voice assistant instructs the user on how to use the system, telling the user to hold down the button to speak, and to say his or her name to be identified.

During driver identification, the system tries to recognize the driver by his or her voice, assigning an identification number (ID) and asking the driver for confirmation. If the identification fails, the system asks if the user wants to start the enrollment of a new driver or perform the identification again.

The new driver enrollment is started if the driver is not identified by the system and chooses to start a new enrollment. In addition, the registration is started automatically by the system if there are no drivers registered yet. During the registration, the system prompts the driver to say his name again, and then to say each command twice. This process takes approximately 1 minute.

The command recognition step is performed after the driver is identified by his or her voice or after the enrollment of a new driver is completed. The system indicates that it is ready to receive commands by a sound warning. After each recorded voice command, the system checks if the voice is really the driver's voice, recognized at the identification or enrollment step. If it is the driver's voice, the command is processed and the vehicle function is activated or deactivated according to the recognized command and the current state of the vehicle function.

The transitions between steps, indicated in the figure 32, mean, respectively:

1. The system has finished the instructions to the user;
2. The driver has confirmed the identity recognized by the system;
3. The driver has chosen to start the new driver enrollment, or the system has acknowledged that no drivers are registered;
4. The new driver enrollment is completed.

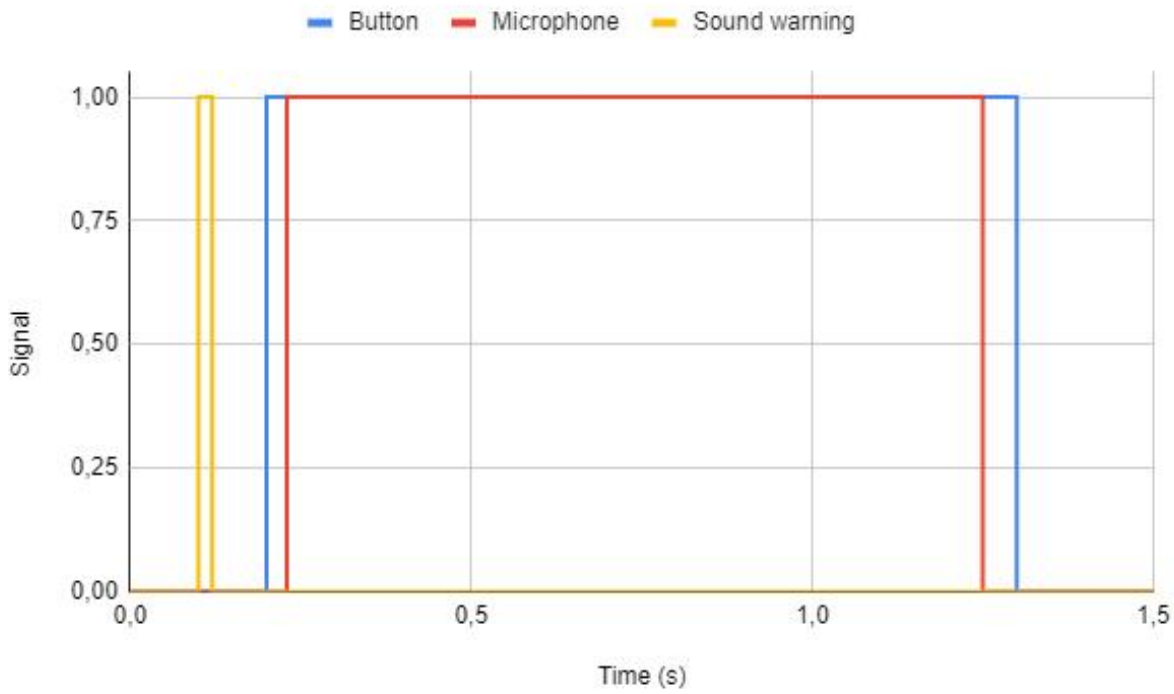
B.3 How to use the system?

The driver identification and new driver enrollment must be performed with the vehicle stopped.

The interaction with the system is done by using a button and a microphone. The user must wait for the sound warning that indicates that the system is ready before pressing the button.

To record audios of commands or the name (for identification), the user must hold the button long enough for the entire audio, as shown in the graph 16.

Graph 16 – Appropriate way to interact with the system.



Furthermore, the button is also used to respond to system requests in the driver identification process.

If the system identifies the voice, the user will be asked whether he/she wants to confirm the recognized identity or perform a new identification. To confirm the identity, the user must hold the button for 1 second. To make a new identification, the user must press the button.

If the system does not identify the voice, the user will be asked if he/she wants to start a new driver enrollment or try the identification again. To start the enrollment of a new driver, the user must hold the button for 1 second. To make a new identification, the user must press the button.