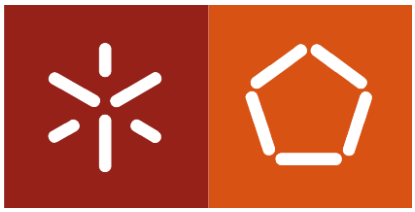


Universidade do Minho
Escola de Engenharia

André Barbosa, PG50216
Carlos Soares, PG50280

Driver Drowsiness Detection (D3)

Master's in Industrial Electronics and Computers Engineering

Embedded Systems

Professor:
Adriano Tavares

September 22, 2022

"Sometimes it is the people no one imagines anything of who do the things that no one can imagine."

Alan Turing

Contents

List of Figures

List of Tables

1 Introduction

It is estimated there are about 1.4 billion drivers in the world, although the number may be even higher, closer to 2 billion. There are three types of distracted driving. Visual distractions, manual distractions, and cognitive distractions [1], but we will focus on visual distractions. Putting away the cell phone, many previous scientific types of research have determined that the main factors causing distraction are drowsiness, fatigue, and driving under the influence. A large-scale international survey asked drivers about driving fatigue. In Europe, about 20 percent of drivers reported that during the past month they had driven at least once when they were so fatigued that they had difficulties keeping their eyes open [2]. In Portugal, a study conducted by the Observatório da Prevenção Rodoviária Portuguesa (OPRP) (Observatory for Portuguese Road Prevention), concludes that fatigue is responsible for 20 percent of the accidents in Portugal, and about 23 percent of drivers have already felt extreme fatigue in the wheel, 3 percent report that they have even fallen asleep, and 2 percent confess that they have already had an accident or narrowly avoided it [3].

The driver's condition, which involves staying focused on the road, is the most important aspect to consider whenever one is driving. Unfortunately, ignoring the importance of this could result in severe physical injuries, deaths, and economic losses. Just for comparison, studies referred by the Autoridade Nacional Segurança Rodoviária (ANSR) (National Road Safety Authority), admit that 17 hours of sleep deprivation corresponds to driving with a blood alcohol level (BAL) of 0.50 g/l and that after 24 hours of sleep deprivation, the decrease in motor performance is equivalent to that observed in individuals with a BAL of 1.0 g/l [4].

1.1 Problem Statement

To overcome this problem, we present Driver Drowsiness Detection (D3), focusing on improving safety for people, reducing road accidents, and decreasing concern for human beings. The core of this concept is eye tracking and blink detection in conjunction with a heart-rate sensor. Thus, the necessary signals are received to alert the driver through the D3's alert system, and then the driver can focus once again on what is most important, on the road and everyone's safety. The D3 is best suited for transport companies, where drivers drive for many hours. Here a company can get more information about the drowsiness of its drivers.

2 Market Research

The Driver Drowsiness Detection System market has witnessed a growth from USD million from 2017 to 2022 with the highest CAGR (Compound Annual Growth Rate) [5]. Is anticipated to rise at a considerable rate during the forecast period, between 2022 and 2029. Technological innovation and advancement will further optimize the performance of the product, enabling it to acquire a wider range of applications in the downstream market. Nowadays, drivers refrain from taking breaks during long trips, thereby increasing the risk of road accidents. Automotive manufacturing companies are working towards developing safety systems to curb the increasing ratio of road fatalities. Moreover, the government is imposing stringent regulations on driver and passenger safety and taking various initiatives to reduce road accidents. It is possible to see that a key market driver is the rapid adoption of advanced technology, the rising safety concerns, and rising popularity are expected to drive the growth of the heads-up display market in the forecast period but on the other hand a restraint is the high cost that is expected to hinder the growth of the heads-up display market. Some of the prominent companies that are present in the global automotive driver drowsiness detection system market include Robert Bosch GmbH, Continental AG, Valeo, Hella GmbH Co. KGaA, Autoliv Inc., Denso Corporation, Magna International Inc., Aisin Seiki Co., Ltd., and TRW Automotive among others [6].

Analyzing Figure 1 we can see that the Drowsiness Monitoring System can grow a lot even in comparison with the other driver assistance systems.

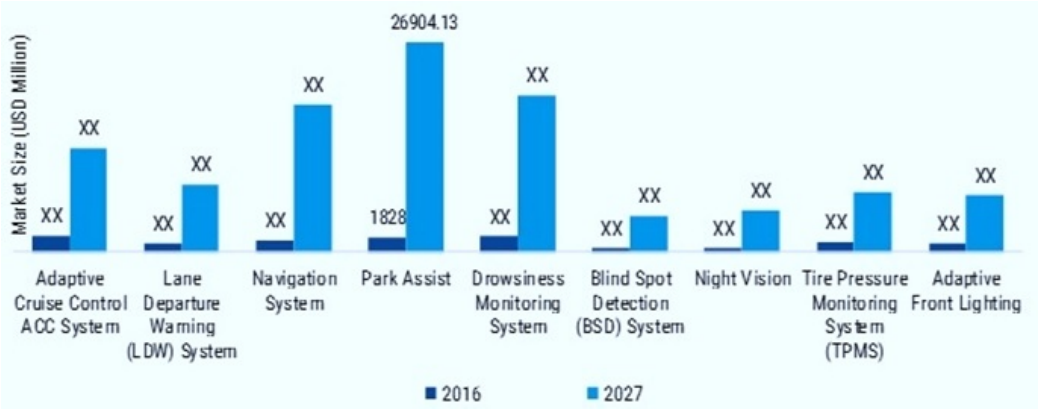


Figure 1: Driver Assistance System (DAS) Market.

2.1 What's on the Market

Cipia-FS10 [7], a similar product to ours, is video telematics and driver monitoring device as you can see in figure 2.2. Uses computer vision and Artificial Intelligence (AI) and with the help of an IR sensor, Cipia-FS10 tracks the driver's head pose, eyelids, and direction of gaze to provide a real-time assessment of the driver's state (attentive, distracted, drowsy). The price of this product is 370$ and we pretend to make our similar product more low-cost.

Figure 2: Cipia-FS10 driver monitoring device.

2.2 Scope

The scope for the future will mainly be focusing on the utilization of outer factors such as vehicle states, sleeping hours, weather conditions, and mechanical data for fatigue measurement.

3 System

3.1 Features

In this section, the features of our system are presented. It is already possible to determine which features the system must have. The following measures have been used widely for monitoring drowsiness: behavioral measures and physiological measures, as shown in Figure 4.

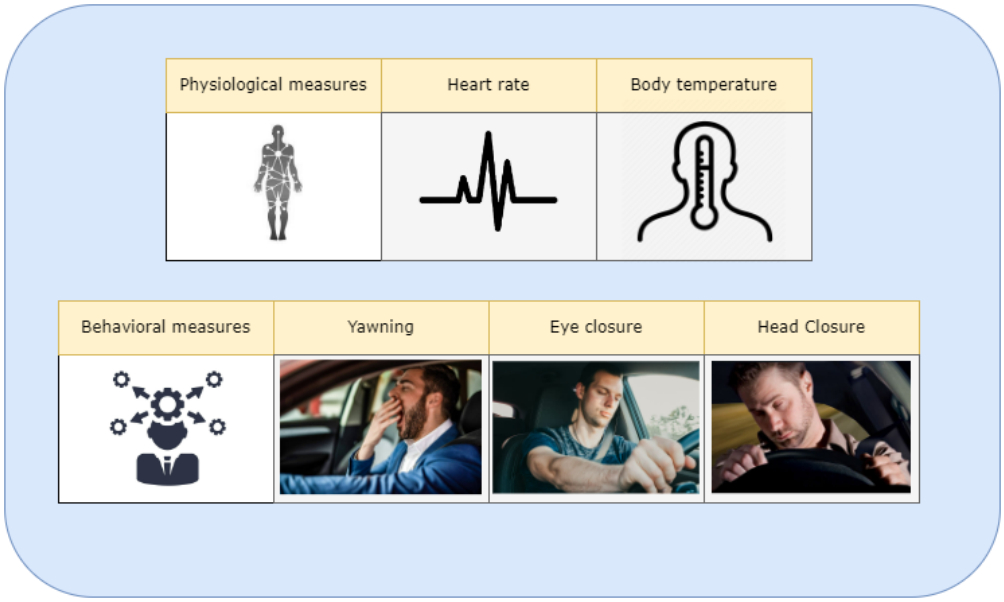


Figure 3: Measures to monitor drowsiness.

Therefore, by analyzing these detection mechanisms it is possible to identify necessary features. Thus, a description of the feature is made and its importance for the fulfillment of the proposed objectives is identified, assigning a priority. Two specific ratings are also included to further evaluate the feature (each rated on a relative scale from a low of 1 to a high of 9): the cost of the product and the functionality and how essential this functionality is to the validation of the system. Finally, a brief description is given of the external behavior of the feature, i.e., how it should react to certain inputs.

Face detection, eyes tracking and state identification

- Description and Priority

This feature shall track the driver's vision and identify whether the eyes are open or closed. This is the most effective method to detect the driver's state of dormancy, and the system must react quickly to these inputs, which were then assigned high priority.

  - Cost: 7
  - Functionality: 9

- Stimulus/Response Sequences

  This feature shall analyze when the user's eyes are closed and if it detects drowsiness, it warns the system.

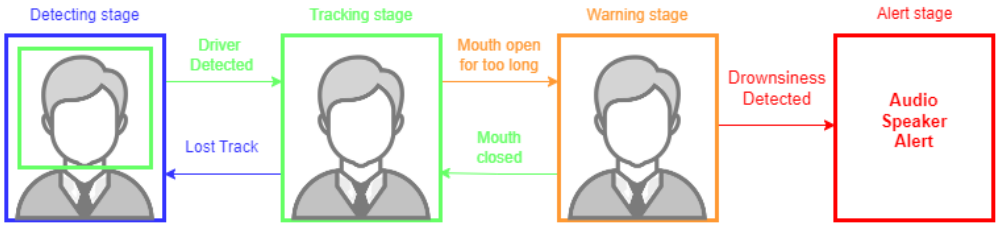


Figure 4: Four Stages of face detection and eyes tracking.

Alert Speaker Analysis

- Description and Priority

  This feature should emit a voice message alerting the driver. These types of messages need to be fast and are essential for system validation, so it was assigned high priority.

  - Cost: 2
  - Functionality: 9

- Stimulus/Response Sequences

  This feature shall send a voice message to the user if drowsy situations are detected by the system.

Monitoring App

- Description and Priority

  This application allows the user to download the data of the trips made during a period of time. The target of this application is business environments in the transportation sector, as it will allow them to manage and analyze their employees' driving data and detect signs of fatigue and drowsiness in their employees.

  - Cost: 1
  - Functionality: 7

- Stimulus/Response Sequences

  Basically, the user will use the application and transfer the data of the last trips made.

3.2 System Requirements and Constraints

This section will present the requirements and constraints identified in the project analysis. On the one hand, we will mention the services that the system should provide, functional and non-functional requirements. Also, will be introduced the constraints on the services or features offered by the system.

3.2.1 Requirements

Functional Requirements

- Collect information to detect drowsiness
- Provide trip data via an application
- Alert the user to obvious cases of drowsiness

Non-Functional Requirements

- A non-intrusive monitoring system that will not distract the driver
- A user-friendly system
- A system that has room for futures upgrades
- Robustness, the system must be tolerant to modest amounts of lighting variations
- Accurate and reliable detection

3.2.2 Constraints

Technical Constraints

- Raspberry pi
- C++
- Buildroot
- Linux
- Device Driver
- Makefiles
- Pthreads

Non-Technical Constraints

- Form a team of two
- Meet the deadline at the end of the semester
- Avoid high costs

3.3 System Overview

After seeing the requirements and constraints we can now analyze the system overview more correctly. At first glance, you can see a general overview of the system where we will have a critical sensor and an actuator.

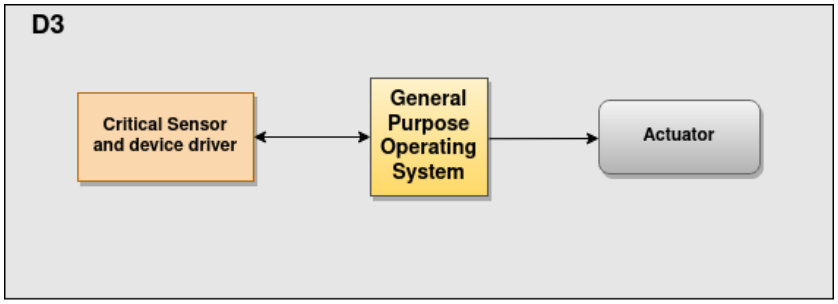


Figure 5: System General Overview Diagram.

Through the system overview diagram, in Figure 6, it is possible to identify the main modules of the system to be developed, and how they interact

and communicate with each other. We have our main local system, with our sensor, Raspberry Pi 4, and our user interface module. Also, has a remote system with a client, the monitoring application, and a server, the database. Finally, a power module for the local system.

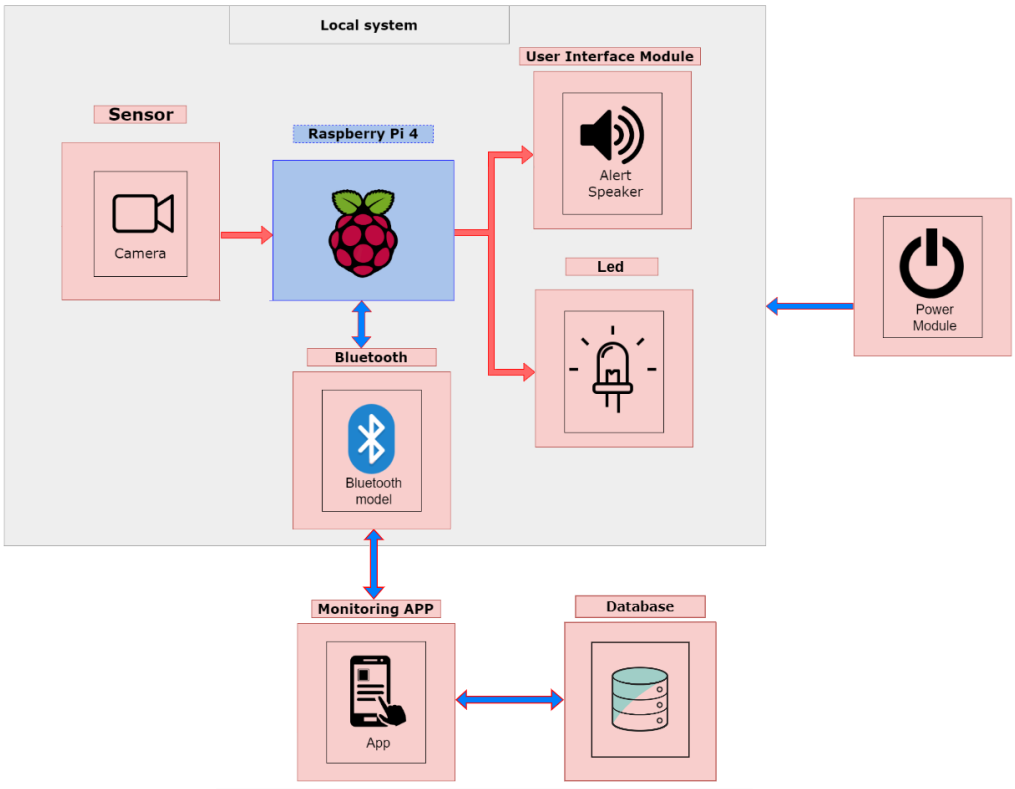


Figure 6: System Overview Diagram.

In the main local system is presented the camera sensor, a microcontroller, and a speaker. When it comes to the sensor, there is a camera for face detection and eye tracking while driving. Also, the processing will be done by the Raspberry Pi 4, where sensor information, can alert the user by activating the alert speaker and can provide essential data by Bluetooth, if necessary. Whenever the driver starts a trip the led lights up.

The remote system is composed of a monitoring app and a database. Thus, making it possible to transfer all the information and later get access via a mobile application. This way the individual user or companies can analyze and be aware of the drowsiness and fatigue status while driving. Also, the monitoring app will be capable of start and end data collection.

3.4 System Architecture

The system architecture can be divided into hardware and software architecture and will be shown in the following.

3.4.1 Hardware Architecture

In Figure 7, one can see the diagram that represents the physical connections of the system. As seen in the development board, Raspberry Pi 4, is the main component in the system, General Purpose Input/Output (GPIO) pins for the Led and Camera Serial Interface (CSI) for the camera and processing it. The communication with the Wi-Fi module and the Bluetooth module is built-in into the Raspberry Pi. The power will come from the power module to all system components.

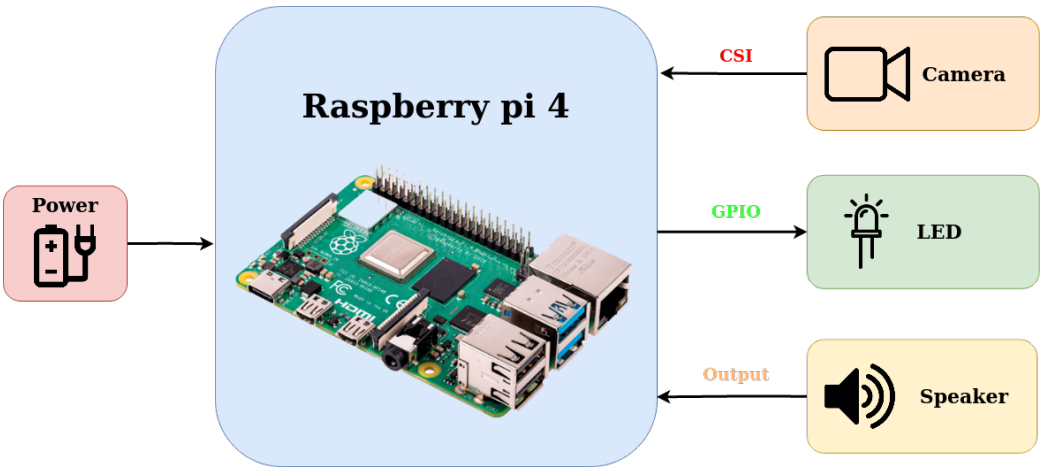


Figure 7: Hardware Architecture.

3.4.2 Software Architecture

In this section, a brief analysis of the expected software system is presented. Three layers are represented: the Operating System, Middleware, and Application Layer.

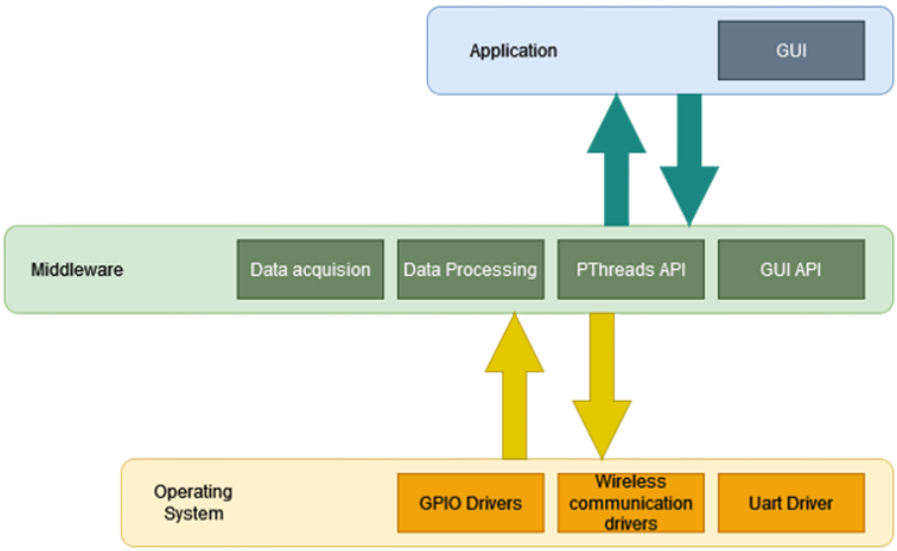


Figure 8: Software Architecture.

4 System Analysis

In this section are presented the conventional diagrams that translate the system behavior.

4.1 Local System

4.1.1 Events

When analyzing a system, one of the most important things, to provide a clear and succinct overview, is to analyze the system's events. Initially, it is identified the event, the response to this event, the source of an event, and what type of event it is, asynchronous, or synchronous. This way, the Table 1 was created for events in local system and Table 2 for remote system.

ID	Event	Response	Source	Type
1	Powering the system on the lighter of the car	Turns ON and OFF the system	User	Asynchronous
3	Sampling the camera	Reads the camera feed	Timer	Synchronous
4	Store data	Stores the values of the camera	Timer	Synchronous
5	Send data	Transfers data via Bluetooth	Timer	Synchronous
6	Alerting User	Activates the sound alert speaker	System	Asynchronous

Table 1: Local System events.

4.1.2 Use Cases

The use cases diagrams allow the developer to describe the various use scenarios (use cases). It expresses what the system should do, without addressing how it should be done. Use cases represent what the customer wants the system to do, that is, the customer's requirements of the system. You can employ usage diagrams to answer the following questions:

1. What is being described? (The system)
2. Who interacts with the system? (The actors)
3. What can the actors do? (The use cases)

The content of a use case diagram expresses the expectations that the customer has of the system to be developed. A basic Use Cases diagram of the Local System is shown in Figure 9.

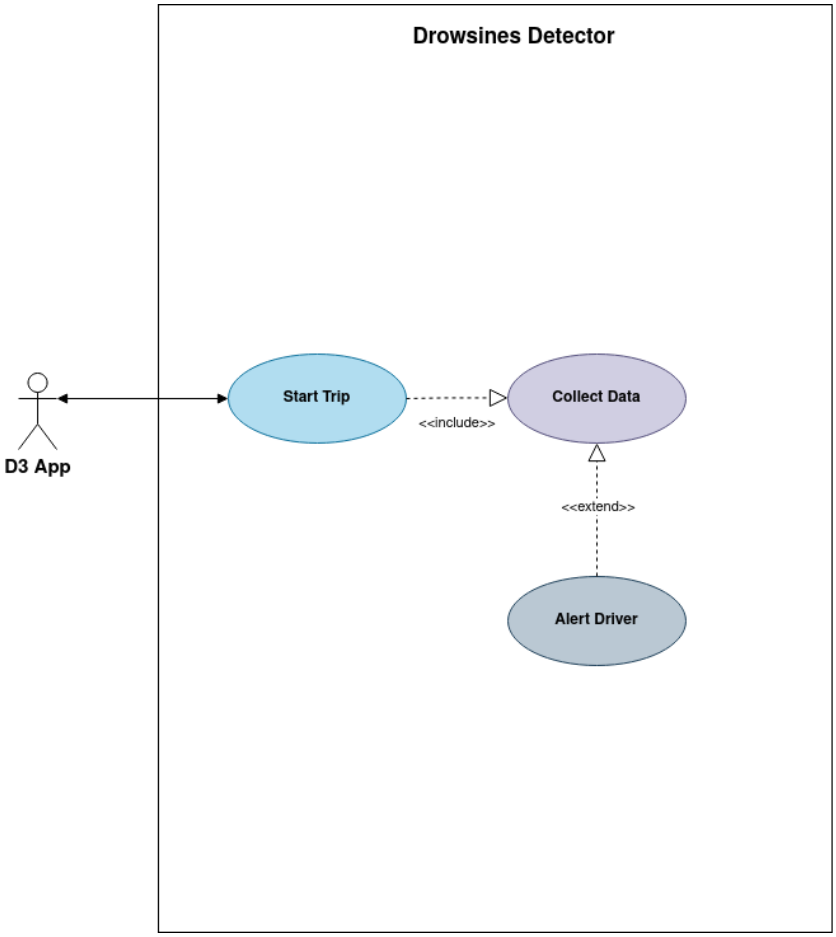


Figure 9: Local System Use Cases.

4.1.3 State Chart

Using a State Chart diagram, one can model the possible states for the system in question, and specify how state transitions occur as a consequence of occurring events. Figure 10 shows the State Chart diagram of the Local System.

When the system is turned on, it will transition to the IDLE state, which will initialize all modules. Afterwards, when the initialization is complete, it will proceed to the Wait Command state, which will wait for the driver to start the trip via the Remote System. In the Process Data state the system will analyze the data collected from the camera and interpret whether an alarm signal should be triggered. The end of the trip will cause the system to proceed to the Sync state, which is responsible for formatting the data

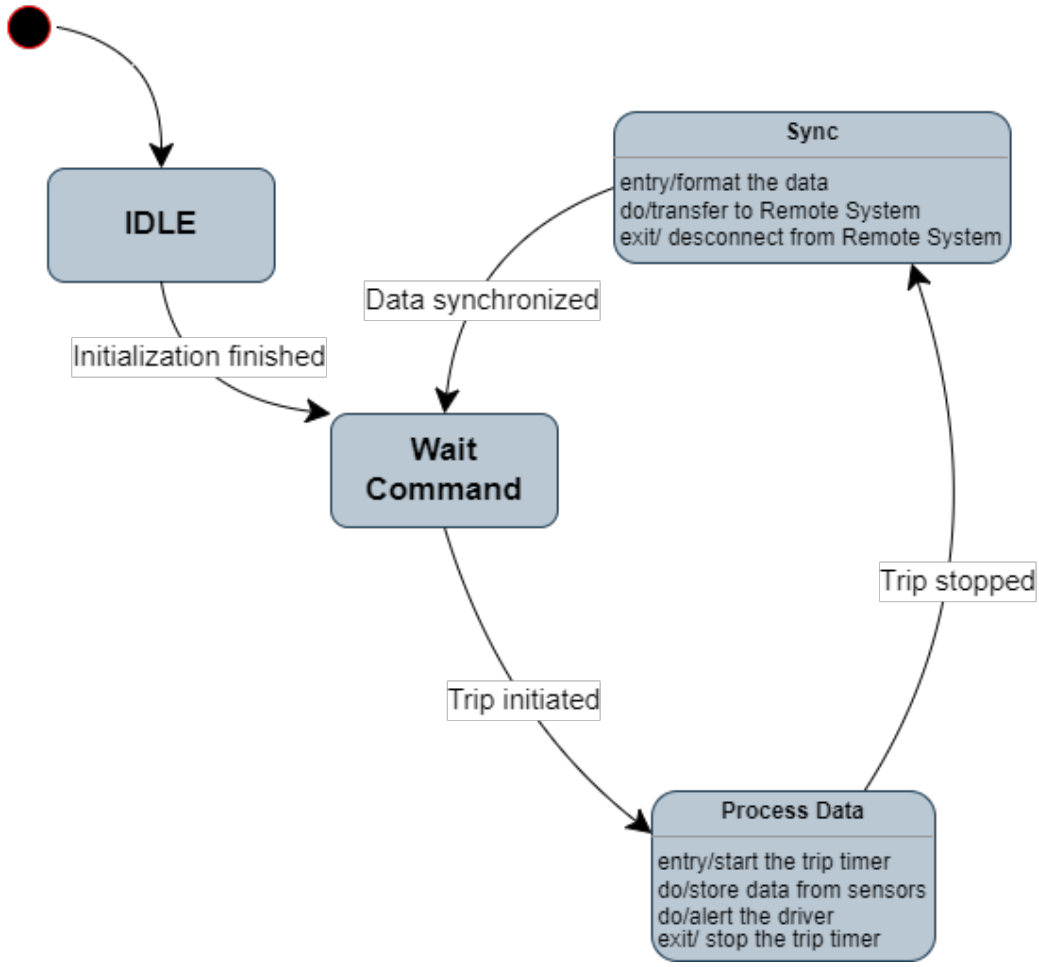


Figure 10: Local System State Chart.

stored during a trip to a predefined format, and then sending the data to the Remote System.

4.1.4 Sequence Diagram

While in the previous chapter only the intra-object behavior was studied, i.e. the life cycle of each object, this chapter will represent the inter-object behavior, i.e. how the objects interact with each other. To do this we will use sequence diagrams. The diagram has been divided into three parts.

In the first diagram, Figure 11, the initial stage of the system is analyzed, where the initialization occur, namely the initialization of the input and output modules. Furthermore, the processing of a command received through the Remote System is demonstrated. If the received command indicates a

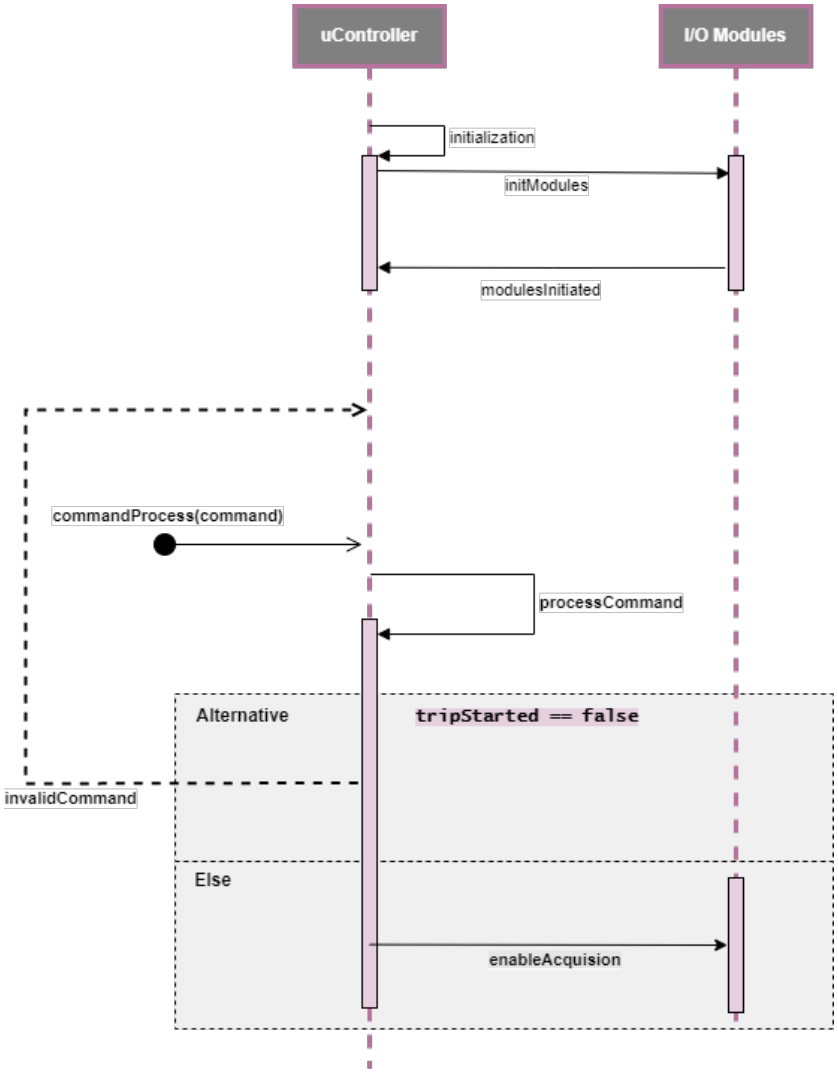


Figure 11: Local System Sequence Diagram 1.

desire to start a new trip by the driver, the system will then enable the acquisition of data received from the camera.

In the second diagram, Figure 12, three partner interactions can be identified: the microcontroller, the camera modules and the sound actuator. This diagram allows a better perception of the data acquisition procedure and the decision making process to alert the driver.

This scheme demonstrates that data acquisition is done repeatedly, the controller issues a data request to the sensor module, which returns the sensor's reading at that given moment. Still, the processing is required on that data, which will trigger a message to the actuator if the value read

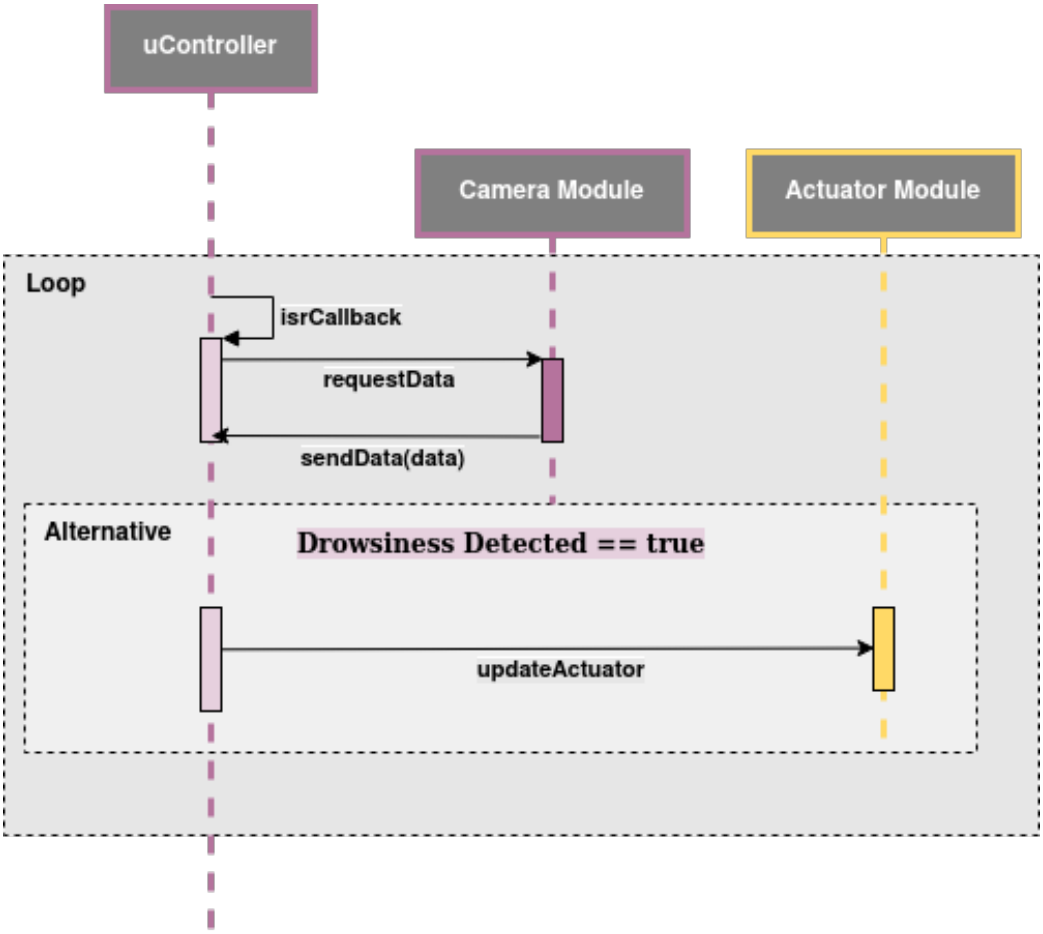


Figure 12: Local System Sequence Diagram 2.

shows signs of drowsiness. If no drowsiness is detected, the system continues its normal execution.

The third diagram shows how the system handles the driver's end-of-trip command, Figure 13.

After processing and identifying the command, if it is identified as a trip terminate command, the data is encapsulated and packaged in a conventional way that is recognized by the Remote System. Once packaged, the data is transmitted to the application, allowing the user to analyze it.

With the analysis of these three diagrams it is now possible to have a better understanding of the system's interactions, as well as its information processing and decision making.

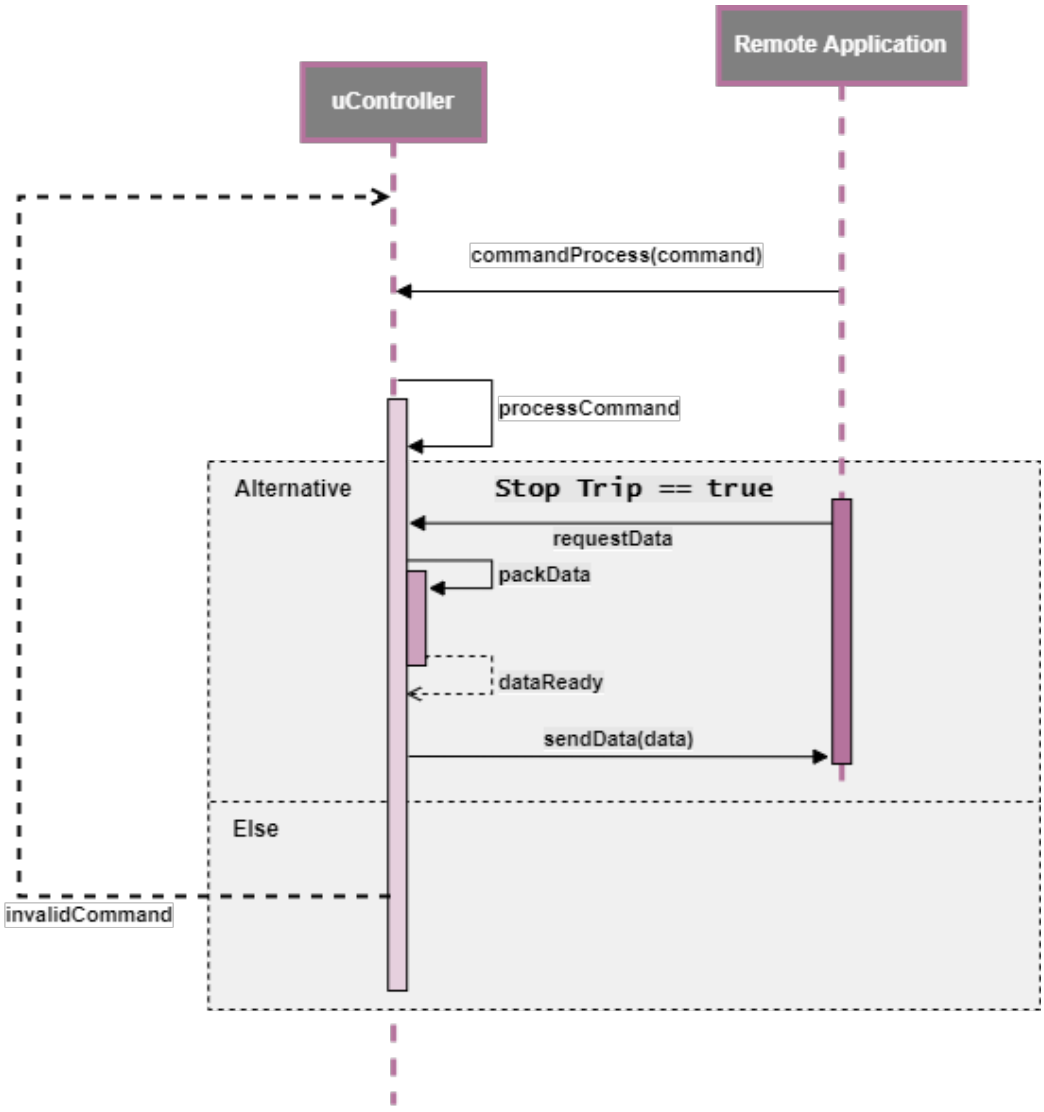


Figure 13: Local System Sequence Diagram 3.

4.2 Remote System

4.2.1 Events

The remote system's events are listed in the table below, Table 2, which are the main events that will affect its behavior in this system. The approach used is very similar to the local system analysis.

ID	Event	Response	Source	Type
1	Create a user account	Creates a new account	User	Asynchronous
2	Establish connection	The remote application will request a connection to the system via Bluetooth	App	Asynchronous
3	Reads data from Bluetooth	Reads the data captured from sensors	App	Asynchronous
4	Login	Login to the application	User	Asynchronous
5	Verify User	Checks if login is valid	User	Asynchronous
6	Sends data via Wi-Fi	Updates the database with new account	App	Asynchronous
7	Read data	Reads from the database the parameters values	App	Synchronous

Table 2: Remote System events.

4.2.2 Use Cases

The content of a use case diagram expresses the expectations that the customer has of the system to be developed. Using a resembling methodology a basic Use Cases diagram of the Remote System is shown in Figure 14.

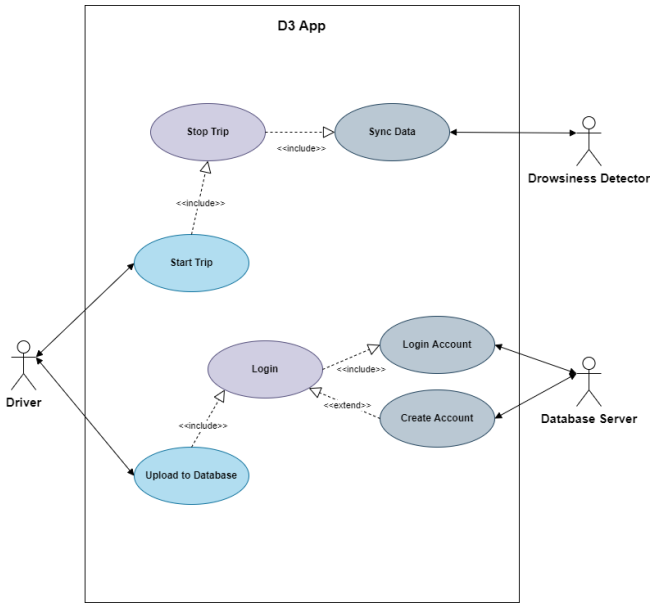


Figure 14: Remote System Use Cases.

4.2.3 State Chart

In a similar way to the Local System, the State Chart Diagram was drawn to model the possible states in this system, as well as to represent the transitions. When the application is started, it will wait for a user selection: start a trip, transfer data to the database, or quit the application, as shown in Figure 15.

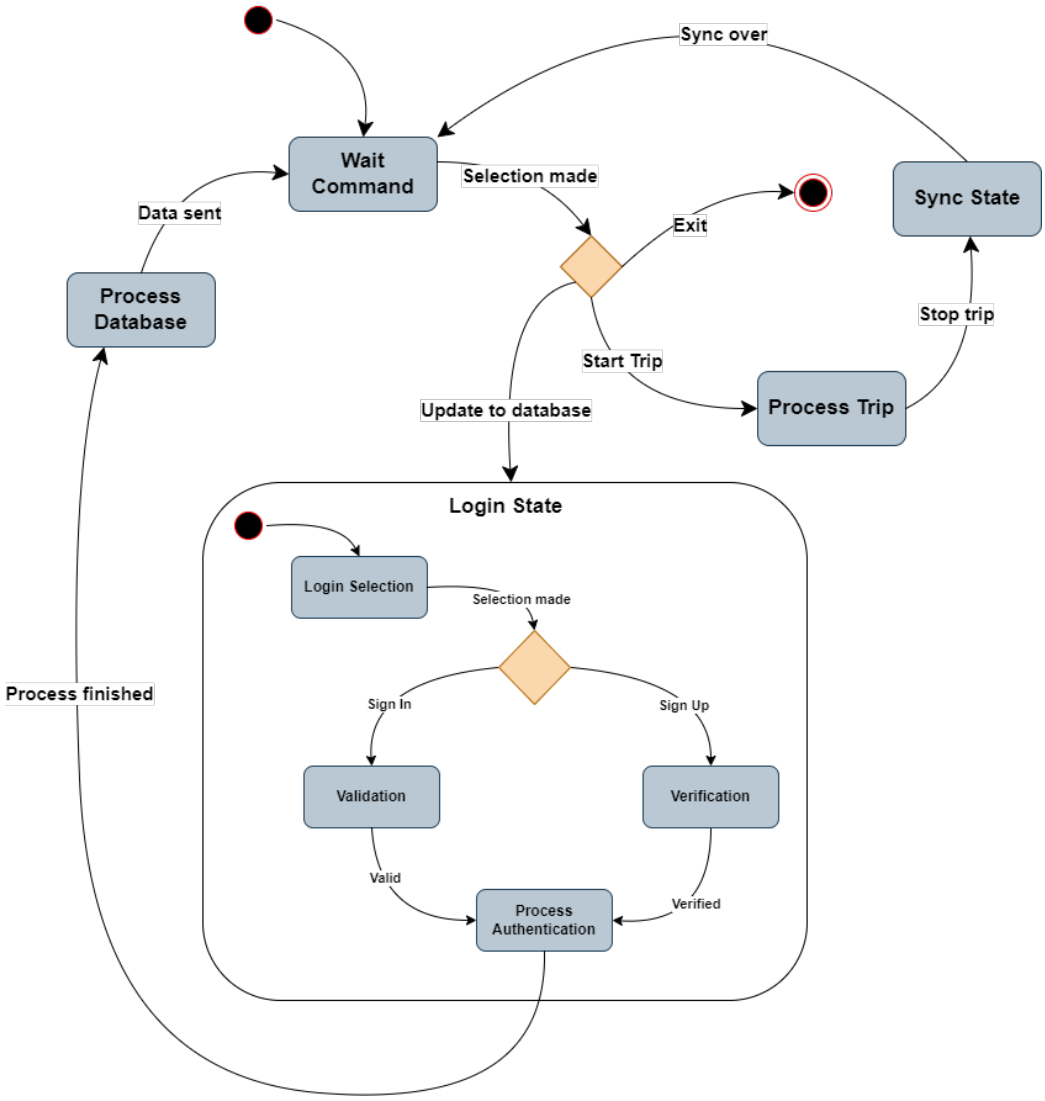


Figure 15: Remote System State C hart.

4.2.4 Sequence Diagram

Figure 16 represents the sequence diagram of the microcontroller with the user, mobile application, and database. Initially, a connection is made between the microcontroller and the mobile application, and actions are triggered by the user, starting with the registration or login operations in the application. It is also possible to observe the Alternative block. If the login is valid, the user has permission to send data. If the login is invalid, the application returns an error to the user.

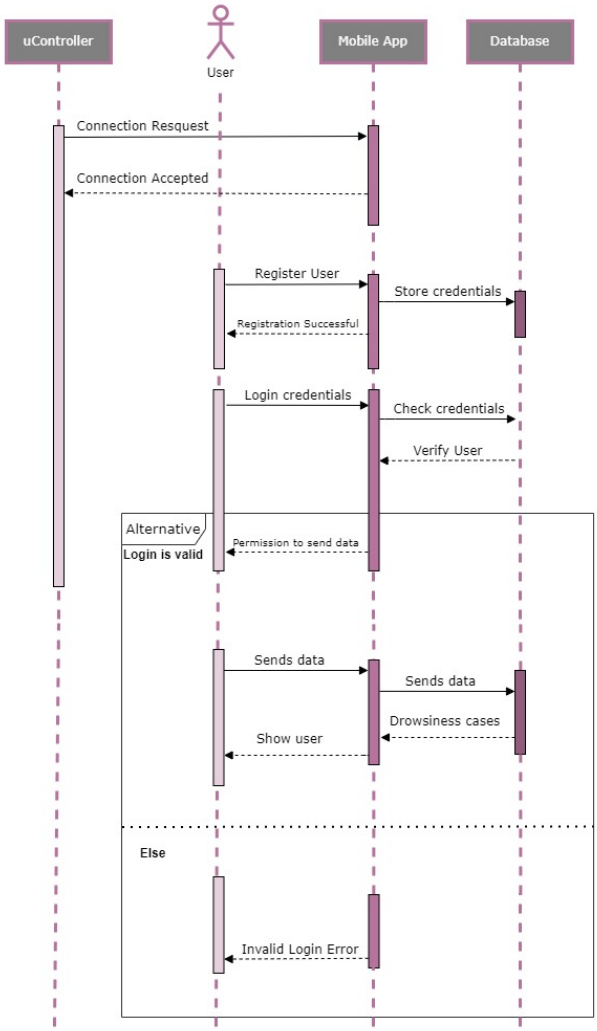


Figure 16: Remote System Sequence Diagram.

4.3 Initial Budget

The Table 3 includes the components, the price of each component, and the total price.

Name	Estimated Cost (€)
Raspberry Pi 4	60
Sensors Module	40
Actuators Module	10
Total	110

Table 3: Initial Budget.

4.4 Task Division: Gantt Diagram.

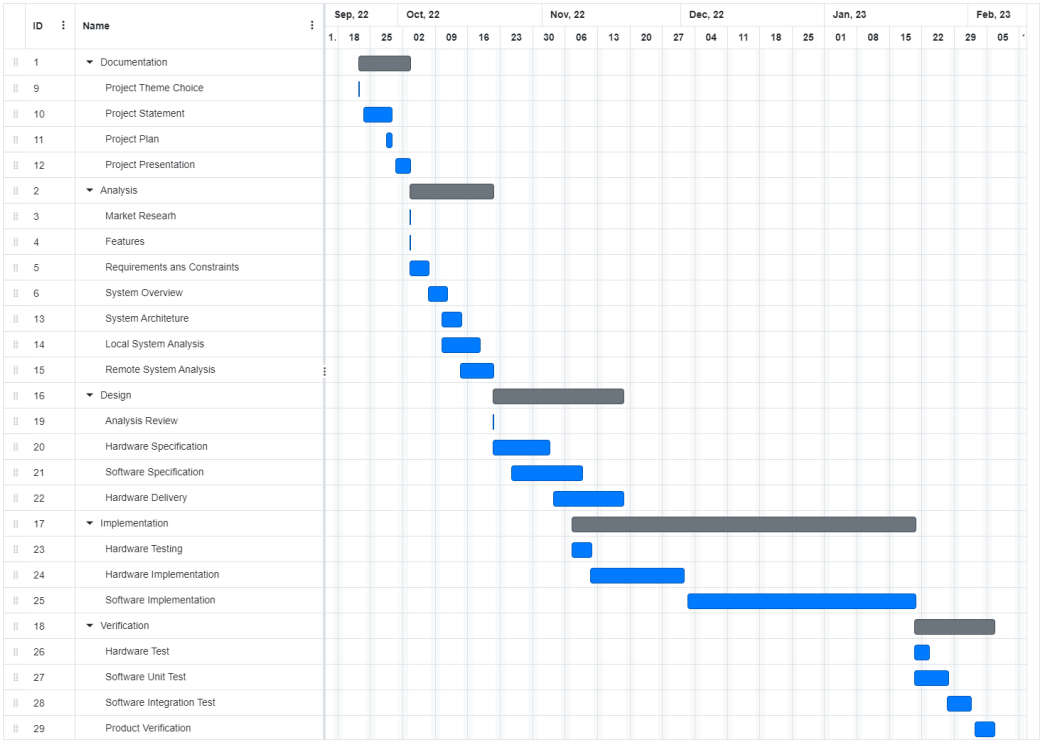


Figure 17: Task Division: Gantt Diagram.

5 Design

5.1 Analysis Review

In the previous chapter was made a brief analysis about the product that is being developed. In a first approach and after a deeply study, the requirements and constraints of the system were defined. In this section, a brief review of the previous step will be conducted, and some concepts and corrections will be presented in order to transition to the design phase.

What's drowsiness?

Throughout the analysis the term drowsiness was mentioned several times, but then what can be understood by drowsiness? Drowsiness is an intermediate state between being completely awake and asleep, which is associated with the desire or inclination to sleep. Drowsiness is associated with slowed reaction times, reduced vigilance, impairment to process information, and also loss of accuracy of short-term memory. One of the mistakes made in the analysis phase, the term fatigue was considered similar to sleepiness, however each term has its own meaning. Fatigue is considered one of the factors that can lead to drowsiness, and is a consequence of prolonged physical exertion. Usually, rest and inactivity relieves fatigue, however, it makes drowsiness stronger.

Factors which influence drowsiness

As is known, there is no instrument that can directly measure a person's sleepiness, so it is necessary to analyze several factors that influence it to estimate a value close to the driver's current state of sleepiness. The main ones are sleep deprivation, working hours (how many hours you work and whether you do night shifts), and sleep disorders. These factors are important in helping to predict symptoms of sleepiness, however, our primary focus is on identifying the onset of sleepiness while driving.

How to measure drowsiness?

Drowsy drivers show several signs, which include repeated yawning, repeatedly departing street lanes and frequent closing of the eyes. Examples of such signs include:

- Yawning;

- Difficulty keeping eyes open;
- Frequent blinking;
- Difficulty concentrating;
- Nodding;
- Swerving out of the lane and delayed reaction to traffic;
- Unjustifiable variations in speed.

It is possible to identify the state of drowsiness using measurements of these various signs. These measures can be divided into four main categories: image-based measures, biological-based measures (using sensors to measure the driver's bio-signals), vehicle-based measures, and finally hybrid-based measures.

These signs start to become more apparent as the degree of drowsiness increases, and can serve as indicators to measure the driver's level of sleepiness. A precise measurement scale for drowsiness levels is necessary, to systematically evaluate stages of drowsiness and facilitate the development of automatic early drowsiness detection systems. There are several scales, however only the Wierwille and Ellsworth scale will be considered. In Figure 18 is represented the scale.

Levels	Verbal Description
1	Not drowsy
2	Slightly drowsy
3	Moderately drowsy
4	Significantly drowsy
5	Extremely drowsy

Figure 18: Wierwille and Ellsworth scale.

5.2 Hardware Specification

The first step for system design is the hardware specification. In this section it will be presented the necessary hardware for the execution of the project, as well as the peripherals, the pinout, mapping and connection layout.

5.2.1 Development Board

The development board used in this project is Raspberry Pi 4 Model 4B, Figure 19, since it was one of the restrictions imposed in the analysis phase. The board has a several features but the main ones for this project are:

- **Processor**: Broadcom BCM2711, Quad Core Cortex-A72 (ARM v8) 64-bit SoC with 1.5GHz;
- **Memory**: 2GB LPDDR4-3200 SDRAM;
- **Connectivity**: 2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE, one Gigabit Ethernet port, two USB 3.0 ports and two USB 2.0 ports;
- **GPIO**: Raspberry Pi standard 40 pin GPIO header;
- **Video and Sound**: two micro-HDMI ports that support up to 4kp60, a 2-lane MIPI DSI display port, a 2-lane MIPI CSI camera port and a 4-pole stereo audio and composite video jack;
- **Multimedia**: H.265 (4Kp60 decode), H.264(1080p60 decode and 1080p30 encode) and OpenGL ES 3.0 graphics;
- **SD card support**: Micro SD card slot for loading operating system and data storage;
- **Input power**: 5V DC via USB-C connector (the minimum 3A), a 5V DC via GPIO header (the minimum 3A) and Power over Ethernet (PoE);
- **Environment**: operation temperature between 0^{o}C and 50^{o}C.

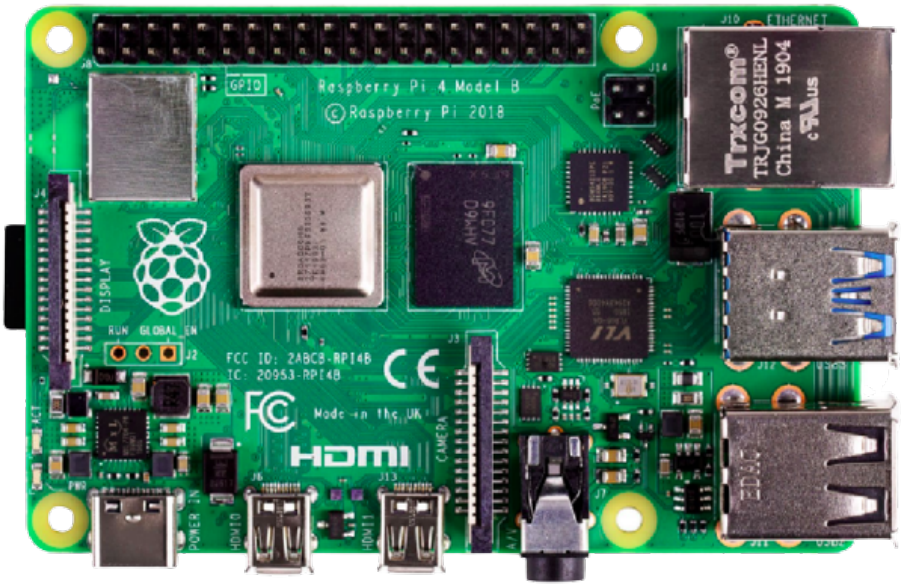


Figure 19: Raspberry Pi 4 Model 4B.

SD Memory Card

This development board supports SD Card, therefore it will be used to store the embedded Linux operating system image. It is used the SanDisk Ultra Memory Card, Figure 20, with 32 GB more then enough to store everything that is necessary to run the system and handle it.

Figure 20: SanDisk Ultra Memory Card.

5.2.2 Camera

This project requires a camera that can work by day as well as by night and since its main purpose is to detect various parts of the human face a good camera is required. Nevertheless needs to be compatible with the board in use, in this case, the Raspberry Pi. The chosen camera is the one as shown in the Figure 21.

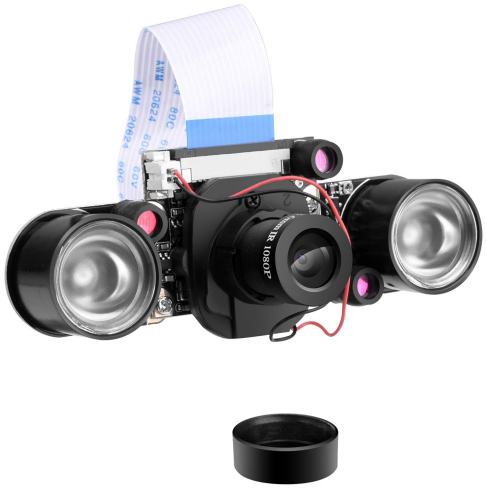


Figure 21: Camera with Night Vision mode and IR filtering.

The reason behind the choice of this particular camera is that it has nigh mode with infrared filtering (IR), meaning that during the night or in low-light environments it is capable of providing a clear image. Its main specifications are:

- **Resolution**: 5MP or 1080p;
- **Connection**: Camera Serial Interface (CSI), 15-pin ribbon cable;
- **Perspective**: 72º;
- **Focal Lenght**: 3.6 mm adjustable;
- **Aperture**: 1.8f;
- **Mode**: Automatic Day and Night modes switchable;
- **Night mode IR Filtering**

Test Cases:

It is important to know how the camera will behave in certain events, thus being able to predict various outputs. In Table 4 are shown test cases to this module.

Test Case	Expected Ouput	Real Output
Video Capturing	Clear output video	-

Table 4: Test Cases: Camera.

5.2.3 Green Led Sensor

Another feature of our system is the ability to, gently show the driver that a trip is in progress with D3 activated, being even more useful when it is dark. This way the user doesn't need to turn on the lights of the car or to be distracted to see if the D3 is actively running. In order to accomplish this, a green LED was chosen, since its a color easy to identify, like the one in Figure 22.

Figure 22: Green Led.

Specifications:

- **Power**: 1.8-2.2VDC forward drop;
- **Max current**: 20mA;
- **Suggested using curren**: 16-18mA;
- **Luminous Intensity**: 150-200mcd;

Pin Configuration:

In Figure 23 is shown the green LED pinout.

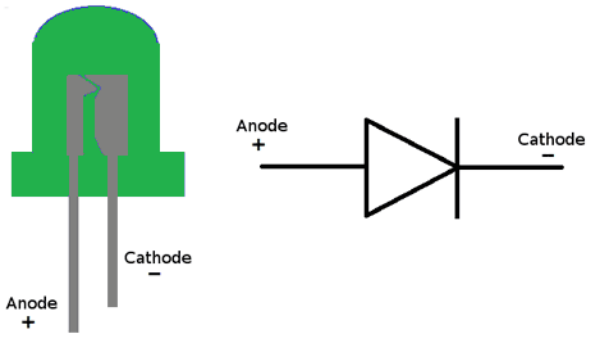


Figure 23: Green LED pinout.

Test Cases:

In Table 5 are shown test cases to this module.

Test Case	Expected Ouput	Real Output
Turn on green LED	Green LED turns on	-
Turn off green LED	Green LED turns off	-

Table 5: Test Cases: Green Led.

Connectivity:

LEDs have a relatively low resistance and if connected directly to the Raspberry Pi's GPIO pin, can draw too much current and damage the pin. Thus, a resistor is used in series with the LED to limit the current flow and protect the Raspberry Pi's GPIO pin. The value of the resistor depends on the forward voltage and forward current of the LED. The resistor used that meets our needs is a 220ohms resistor, as the one in the Figure 24.

Figure 24: Resistor 220 Ohms.

Specifications:

- **Value**: 220 Ohms;
- **Tolerance**: 5%;
- **Power**: 1/4W;

5.2.4 Alert Speaker

The alert speakers represent an important part in the project, since it is from them that the driver is able to wake up from every time that the system detected drowsiness. In Figure 25 it is possible to see which speaker was chosen for the project.

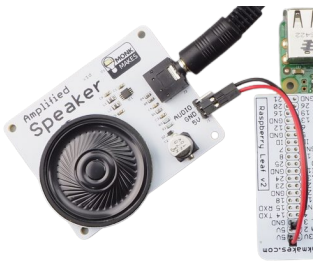


Figure 25: SKU00077 Amplified Speaker Kit for Raspberry Pi.

Since the raspberry pi has a has a 3.5mm jack, which is compatible with the connector of the speakers then the connection will be made between them.

5.2.5 Power Module

As specified before, it's needed a power module to power the system. Since the system will be used in cars it makes perfect sense to use the car cigarette lighter socket. For this it is necessary to make sure that it will provide a solid 3A and maintain 5V at that current level. With this in mind the choice fell of on using a Car Cigarette Lighter Adapter plus a USB to USB type C cable.

In the Figure 26 below it is possible to see the power supply chosen for the project and then its specifications.

Figure 26: Power Module.

Specifications

- **Input**: 12V - 30V;
- **Output**: 5V - 6A / 9V - 2A / 12V - 1.6A;
- **Power**: 30W;
- **Multiple protection**.

5.2.6 Hardware Connection

In this section, the system hardware connections between the hardware modules described in the previous section will be demonstrated as its possible to see in the Figure 27.

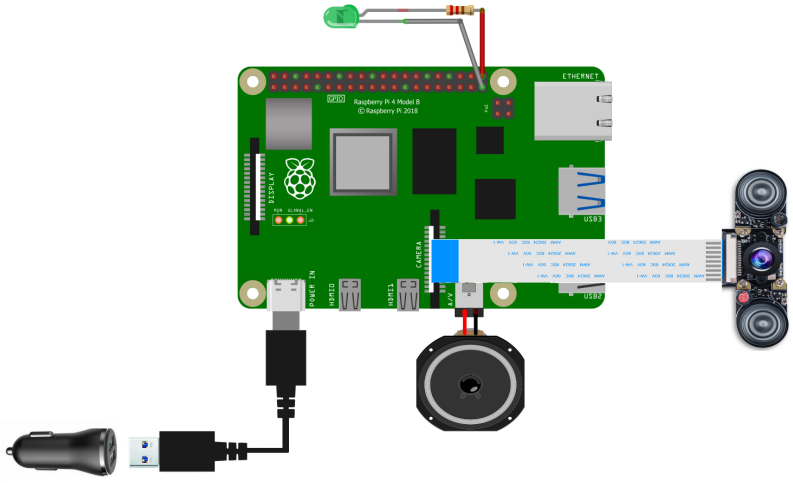


Figure 27: Connections Layout.

5.3 Tools and COTS

During the development of the project, it will be necessary to use some tools and commercial off-the-shelf (COTS) applications that support the implementation of the system. The choice of these tools is important in facilitating and helping the work of the project developers. This subsection will specify and explore which tools and COTS to use and what they consist of.

Tools

All the tools used during the design phase which will be used in the implementation phase will be identified and described.

- **Git**: free and open source distributed version control system;
- **GitHub**: provider of Internet hosting for software development and version control using Git;
- **Buildroot**: tool to configure and generate the Raspberry Pi Kernel image;
- **C/C++**: oriented programming language, used to develop local system and remote system core;
- **Visual Studio Code**: Code editor;
- **Makefile:** Used to compile the developed programs;

- **CMakeList:** Used to compile OpenCV algorithms;
- **Draw.io**: free diagramming application that allows users to create and share diagrams within a web browser;
- **MySQL**: Relational database management system used for the remote server database;
- **Fritzing**: this program will be used to draw the wiring diagram of the components used;
- **Fusion 360**: for 3D print designs.
- **PhPMyAdmin**: for managing and administrating MySQL databases.
- **Postman**: HTTP client tool used for testing and developing APIs.

COTS

The COTS of D3 will be presented. Commercial-Off-The-Shelf Software (COTS) is software that is commercially produced and sold, ready to use without any form of modification by the user, and accessible to everyone. Are solutions which are then adapted to satisfy the needs of the product and to abstract the developer of some parts of the software implementation.

- **POSIX Threads API**: Used for thread creation and management;
- **OpenCV API**: Used for image capture and processing;
- **RaspiCam API**: C++ API for using Raspberry camera with/without OpenCv;
- **BlueZ**: provides a complete software solution for Bluetooth communication on Linux systems;
- **ALSA**: The Advanced Linux Sound Architecture (ALSA) provides audio and MIDI functionality to the Linux operating system. Pulse Audio and Jack operate over ALSA;

5.4 Software Specification

This section focuses in more detail on specifying the various software subsystems to be designed. This specification will take into account all the constraints and requirements discussed above. To clarify the design adopted for the system we will follow a top-down approach, i.e. first we will study several aspects that concern the system as a whole, and then define small subsystems that will make the design phase easier and more straightforward. A well-documented design phase will make the later implementation phase much easier. If this is not the case, issues may arise in the future that makes maintenance and continued code support extremely difficult. Among many, some problems can be highlighted:

- Lost of integrity;
- Lost is original simple and "clean" structure, becoming difficult to understand by the developers;
- Difficulties in promoting changes in the code.

When addressing these concepts it is important to remember, the need for good structuring and readability of the program code, as will be emphasized later in the implementation phase. The preparation of these projects in business environments may contain several development teams, so adopting this discipline will be significantly reduced time losses for code interpretation, as well as facilitate the other team members to analyze and identify possible future errors.

In conclusion, this section will define how the study performed in the analysis chapter, as well as the specified requirements and constraints, will be translated into the final software implementation of the product.

5.4.1 Drowsiness Detection

As the information presented in the review analysis, an algorithm can now be implemented that can reliably identify the driver's state of drowsiness. Due to time constraints, to meet the final deadline it was decided to choose the metric called eye aspect ratio (EAR) to measure the drowsiness level, which does not require intensive computer processing, to evaluate the blinks, i.e. to identify the blinks as a signal that will lead to a measure of driver drowsiness.

Drowsiness Detection Algorithm

After several searches and always with the constraints present, it was decided to implement a drowsiness detection mechanism shown in Figure 28.

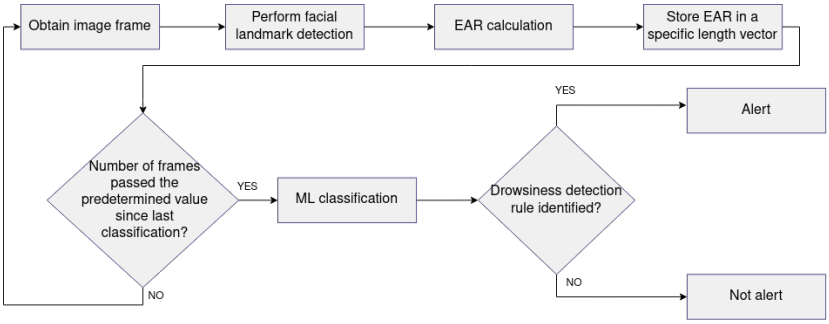


Figure 28: Drowsiness detection methodology proposed

The proposed methodology can be divided into three main parts: eye detection, EAR calculation and blink classification, and real-time drowsiness detection. In more detail, using the camera specified in the hardware specifications, you will get a frame. This frame will go through an algorithm that will identify the facial landmark, especially the eyes landmark. With the use of this algorithm we obtain points similar to the ones represented in Figure 29.

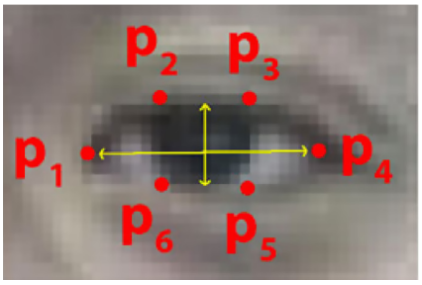


Figure 29: Eye detection by algorithm

From there it becomes possible to estimate the EAR value for the given frame using the formula in Figure 30.

Then the calculated value will be stored in an array with a predefined length. After reviewing some studies, the length was set to 15 values [12]. When a frame is put into the array the value that was recorded the longest is taken out. This leaves us with an array of the last 15 recorded EAR values.

It was analyzed that a blink only occurs in the middle five frames of the collected set of 15 frames (6th to 10th frame), with this procedure it

$$EAR = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$

Figure 30: EAR estimation value

is avoided that the same blink is computed twice during the detection of the sleepiness state, since the designed model selects inputs every 5 frames. Thus, when the collection of a frame is registered, it is checked whether the number of frames collected since the last classification has exceeded the value 5. If so, the classification of the 15 EAR values is then performed.

Finally, these 15 selected values will be used as input for classifying the set of frames. For this, Machine Learning (ML) will be used to determine whether the set of frames corresponds to open eye, short blink or long blink, respectively the values of 0, 1 and 2.

In a study analyzed [12], the possibility of using several ML models was verified, and the efficiency corresponding to each model. Thus, based on this study it was decided to use the support vector machine (SVM) model, since it showed more efficiency in determining the state of the blink in the 15 frames. So this model will take an input of 15 values and then determine an output between 0 and 2.

Detecting one output of long blink is not enough to represent drowsiness. More information is required to make inferences about drowsiness. So we defined some rules based on related literature [13], to define drowsiness:

- Rule 1: If within a period of 60 seconds, the proportion between the '2' output (long blinks) and the sum of '1' and '2' outputs is higher than 25%.
- Rule 2: If 5 or more consecutive outputs (SVM predictions) are 2 (long blink). This happens when the user stays with their eyes closed during face tracking.

In conclusion, if any of these rules hold true, it can be safely assured that a state of drowsiness has been recorded, and the driver should be alerted quickly.

Machine Learning Algorithm

As specified above the method used will be the SVM one. The SVM model is a supervised learning model, which will analyze data and classify that data into categories, which in our case will be open eye, short blink and long

blink. SVM uses a set of parameters necessary for data classification, these being C, the penalization factor for wrongly classified data, and λ, related to the localization and flexibility for the Radial Basis Function kernel (RBF). After these parameters have been determined, a training will be conducted that will allow the implemented model to more accurately specify cases of drowsiness for various people.

5.4.2 General System

Recalling the analysis performed, the system will have to comply with certain restrictions when it is implemented in software. Thus, it is extremely important that the system be designed based on multitasking, more precisely on preemptive multitasking.

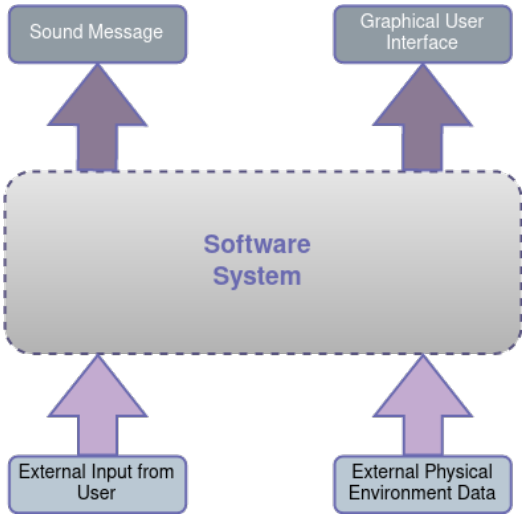


Figure 31: Software System Overview

As stated in the previous chapter, first a more detailed overview of the overall system is demonstrated in Figure 31, as well as its interactions with the external environment. Thus, as discussed in the Analysis section, the external environment will interact with the Software System in two different ways: through external data received by the sensors, and through user input on a Remote Application. Thus it was decided to separate the system into three subsystems, which will deal with different functionalities of the overall system:

- Drowsiness Detection Software System;
- Remote Software System;

- Database System.

In Figure 32, the representation of these subsystems can be seen, and how they interact with each other.

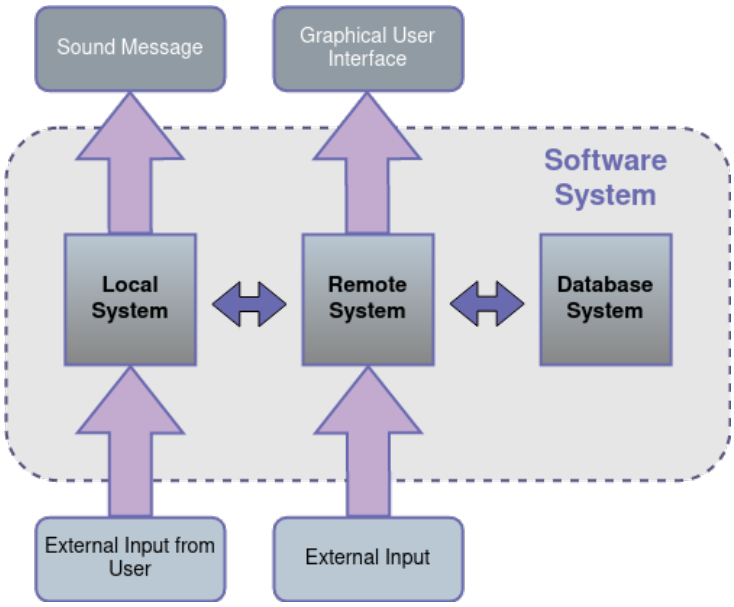


Figure 32: Software System

In this manner, it becomes necessary to specify the design of each subsystem, as well as the interaction between them.

5.4.3 Local Software System

Starting with the Local System, the software designed will essentially have to deal with the reading of data by the sensors, their interpretation, and the decision algorithm that allows alerting the driver in case of drowsiness. In addition, this program will interpret commands received by the Remote System, and will synchronize data collected during the trip with it. The communication between these two programs is done via Bluetooth, so it is necessary to design an communication capable of satisfying the sending and collection of data in a clear and conventional way for each program. Furthermore it will be discussed the class diagrams and flow charts as well as tasks division and priorities of the local system.

Class Diagrams

The use of class diagrams, allows a better visualization of the static structure of the system, that is, it allows you to perceive the constituent elements of the system as well as the relationships between them, however these elements and their relationships do not change over time, they are fixed. One of the advantages of this diagram is that when working in teams, it is easily interpreted and recognized by any developer, since it is one of the most used and flexible platforms for software design.

In Figure 33, the LocalSystem class diagram is represented.

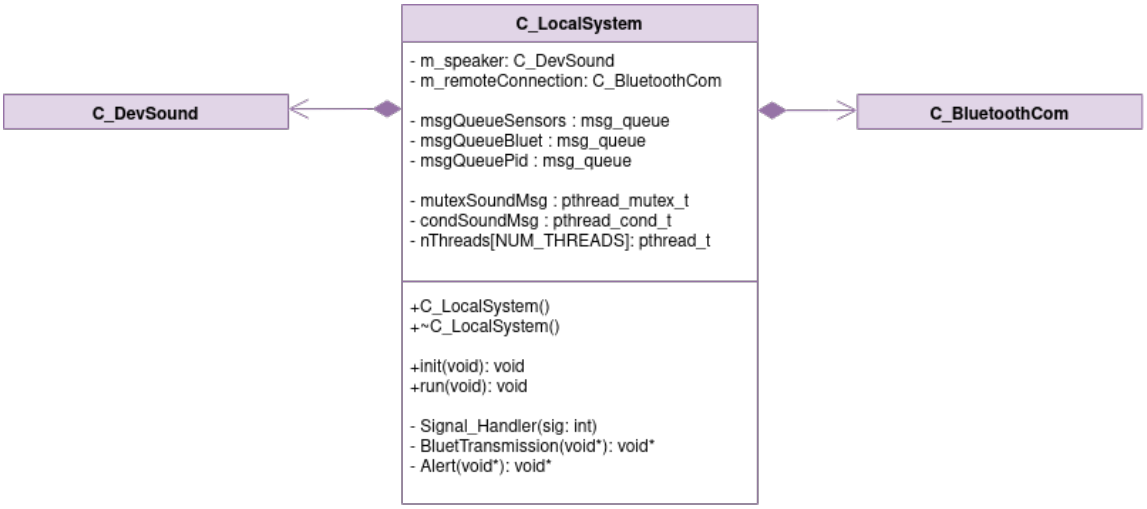


Figure 33: Local System Class

This class has objects from the C_DevSound and C_BluetoothCom classes, which manage the information related to sound processing and bluetooth communication, respectively. There are also two member functions BlueTransmission and Alert, which are associated with the behavior of the class threads. The init function will initialize the attributes and synchronization mechanisms used in the threads, and the run function will create the threads and join them.

In the next Figure 34, one can see in more detail the representation of the C_BluetoothCom and C_DevSound classes.

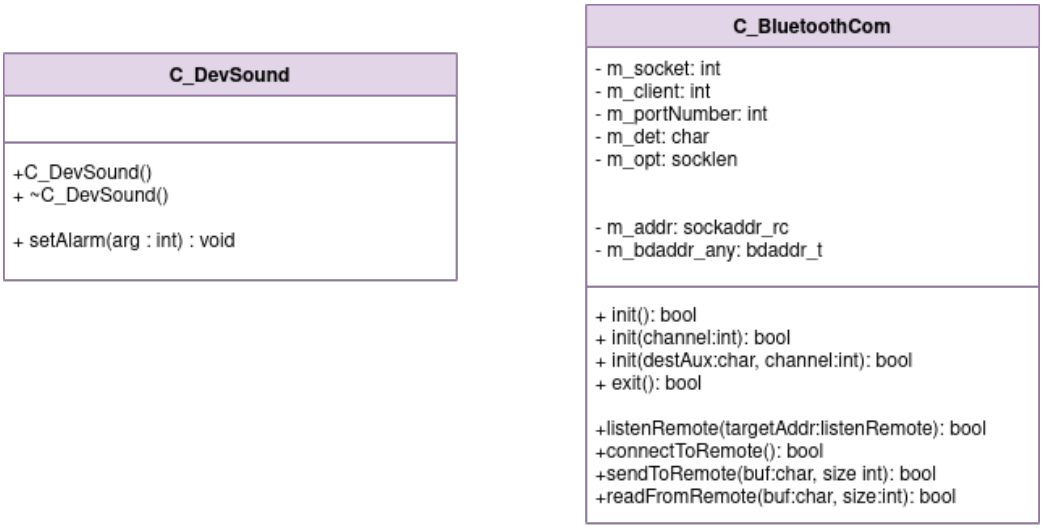


Figure 34: Bluetooth Communication and Sound Device Classes

One can now see the Daemon class diagram in Figure 35. As specified before, it has 1 object to manage data collection, m_camera. The m_drowCam object allows you to perform image processing operations, thus obtaining the drowsiness level recorded in a video frame, and of course the id objects for each of the threads that will be implemented. Regarding the member-functions, it has 2 functions init and run, with the same behavior as the C_LocalSystem class, and 3 functions associated with threads, that will be analyzed in more detail in the flowcharts section.

In the next Figures 36 and 37 you can see the implementation of the classes associated with the C_Daemon class. The C_BlutoothCom class follows a communication protocol very similar to TCP/IP, providing functionality to connect to the target address and send or receive information from it. The C_DrowsinessCam class takes on a very important role in processing the frame captured by the camera. This class gives the programmer a function to process the frame, i.e. to obtain the parameters needed to get the sleepiness level, calculated by the member-function calculateDrowLevel.

As described above, now is possible to demonstrate the implementation of the class related to the camera device, the C_DevCamera class.

The C_DevCamera class will have an OpenCV library object that is called m_camDev and essentially a function to start and stop the camera. The m_camName will be update by the constructor, and should have the same name as the camera device in the OS file system. Finally, one frame can be captured using the member function frameCapture, and will be store in the data member variable named m_frame.

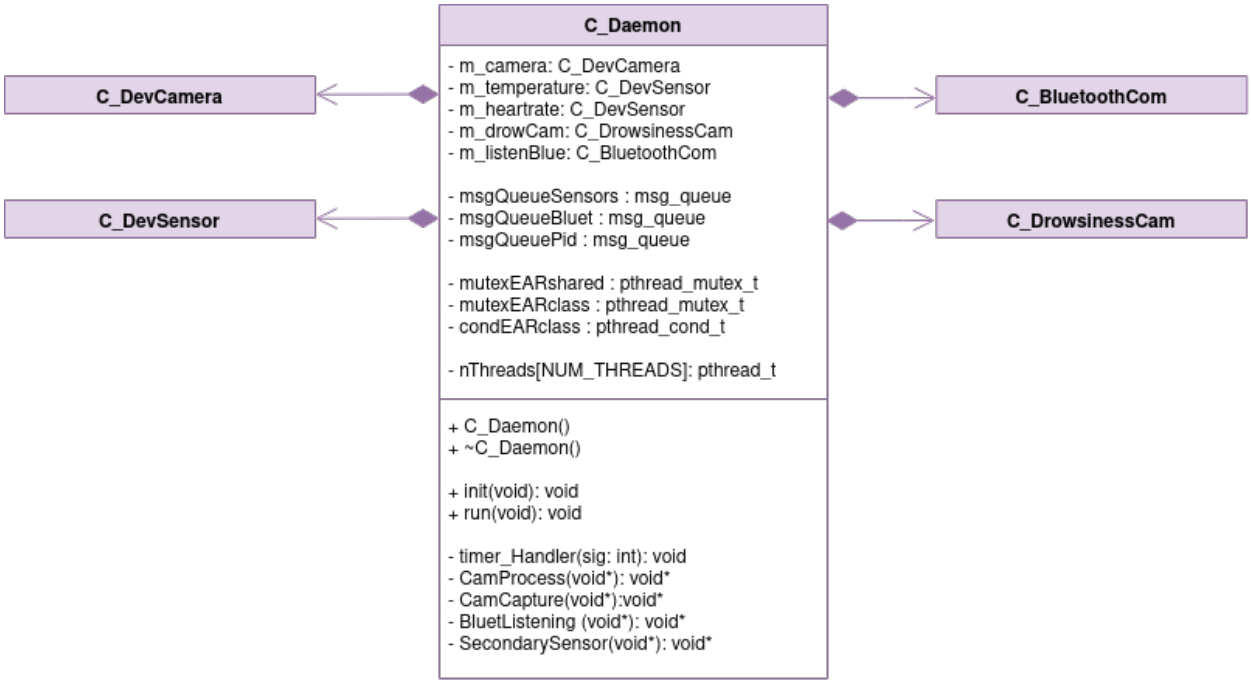


Figure 35: Daemon Class

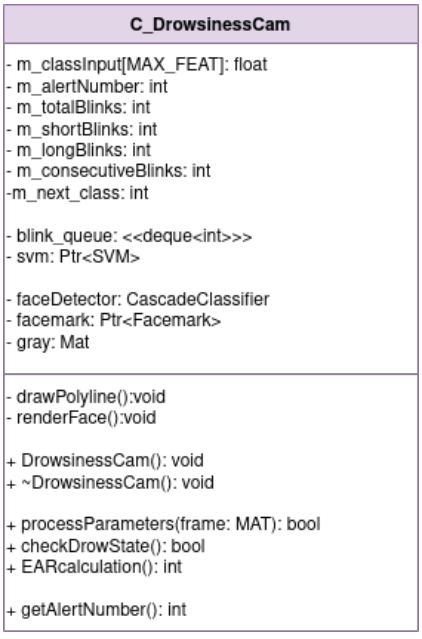


Figure 36: Drowsinesss Calculation Class

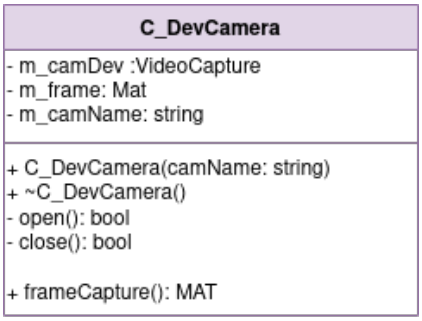


Figure 37: Camera Device Class

Device Driver

Next the implementation of the device driver needed for the system will be specified. The system will have a green LED, being capable to show the driver that a trip is in progress with D3 activated. This GPIO device driver will be implemented as character device driver. Will be a type of driver that controls the behavior of the device connected to the GPIO pins.The necessary functions for controlling the state (on/off) of a LED that is connected to the GPIO, available at the application level, are:

- **Open function:** This function is called when the device is opened for the first time. It initializes the necessary resources for the device, such as allocating memory for device-specific data structures, setting the direction of the GPIO pin as output, and configuring the device.
- **Close function:** This function is called when the device is closed. It releases any resources that were acquired during the open function, such as freeing memory, disabling the GPIO pin, and releasing any locks or semaphores.
- **Read function:** This function is called when the device is read from. In the case of an LED device, this function can be used to determine the state of the LED (on or off). It returns the value of the GPIO pin, which can be either 0 or 1.
- **Write function:** This function is called when data is written to the device. In the case of an LED device, this function can be used to turn the LED on or off by setting the value of the GPIO pin to 1 or 0, respectively.

These functions allow the software to interact with the LED device and control its behavior.

The GPFSELx register will be used to define the operation of the GPIO pins, input or output. The GPFSETx / GPFCLRx are registers that will be used to set or clear pins, respectively. Each register GPFSELx contains 10 GPIOs. As defined in the Hardware Specification section, the green LED will have its output connected to the pin 21 and ground. The register to define pins as input is GPFSEL2[10].

Task Division

Since this is a multitasking system, a clear representation of all the processes and threads that make up the system cannot be missed. The use of multitasking must be carefully planned and studied, otherwise it may present excessive overhead. A good way to reduce overhead is to decrease the number of context switches. To do this, two techniques can be adopted: reduce the number of tasks and avoid different priorities whenever possible. It is important to remember that the execution order of threads is nondeterministic and controlled by the OS scheduler, so it is important to keep in mind that you should never assume a certain order of execution.

So the first thing to do is to identify activities that can be performed simultaneously. To do this we used some criteria:

- Identify different types of processing that can be run independently of each other;
- Identify the computational hotspots that account for the most execution time;

As shown in the analysis, the system will interpret commands received from the remote application, analyze the values received by the sensors, and depending if the driver is awake or drowsy, the speaker will alert the driver.

In more detail, the system will simultaneously read two sensors and a camera. It will process each video frame and identify parameters needed to determine the driver's level of drowsiness. An alternative processing will occur with the remaining two sensors, but this will occur with a lower sampling frequency, in which the value of each sensor will be recorded at a given instant of time, and a processing will be performed to determine if this value is within the normal values of the driver.

For the determination of the level of sleepiness an algorithm will be used that will receive as inputs the parameters received through the collection

of sensors. This processing should occur in parallel with the reading of the sensors, so it is necessary to ensure that the parameters are not modified during the execution of the algorithm, and to ensure that the data collected during the execution of the algorithm is not lost.

Furthermore, a communication will be established with the remote system, and then a computation will take place that will interpret received data and proceed to send data, namely, sending the level of drowsiness, the number of hours of driving, the number of warnings and advice occurred.

Initially, it was decided to split the project between:

- **Main process:** will be used to send the processed data to a message queue that the daemon will read from, define the signal handler and be responsible for handling all the threads, inputs, outputs, and handlers. Also, it will be able to interpret a signal received by the Daemon, if there is one, alerting the driver in case of need.

- **Daemon process:** a Daemon is nothing more than a background process that runs regardless of user input and provides some service. The created Daemon will read the processed data from the camera and continuously look for any cases of drowsiness. If it detects drowsiness, then a signal is sent to the thread alert, in the main process, to alert the driver.

Thus, it is already possible to define the various threads present in the application's execution:

- **Main thread:** will be responsible for starting the program. In this thread all the threads of the application will be created, verifying their correct creation, and establishing the priorities that will be defined in the future;

- **LocalSystem::T_BlueTransmission:** transmits to the remote system, via Bluetooth, the the number of times the alert was activated, in other words, the number of times that was detected that the person driving was drowsy.

- **LocalSystem::T_Alert:** An alert signal received by the daemon will cause this thread to resume its execution and perform the necessary sound processing to alert the driver.

- **Daemon::T_BlueListening:** will receive commands from the remote system, via Bluetooth, to initiate the trip.

- **Daemon::T_CamCapture:** This thread will perform constant frame captures. When capturing a frame, an algorithm will be executed to obtain points around the eye. With the position of these points it is possible to calculate the EAR of this frame. This value will be updated in a shared memory, and if the number of frames obtained since the last classification is exceeded, the thread T_CamProcess will be flagged.

- **Daemon::T_CamProcess:** This thread will analyze the N frame captures that are in shared memory, whenever it is flagged. These values will go through a machine learning algorithm, where it will be returned if in that set of frames there was a short blink, long blink or the eye is open. This value will be stored and will go through two rules that conclude whether it needs to alert the driver or not.

Next will show in more detail how the synchronization between all the defined threads will happen, as well as the rationale for setting priorities for each thread.

Task Synchronization

In a multitasking system, resources and services may be shared among the various tasks, generating critical sections. Therefore, these interactions must be synchronized to predict race conditions, which are nothing more than unexpected results when the timing of threads impacts other threads. Sharing global process data requires more use of the synchronization objects used for controlling access to shared resources[9]. All the values are only modified at the end of the trip, so there is no risk of these values being modified during transmission, and there is no need to use mechanisms to protect this data. On the other hand, the values transmitted in the thread during communication will be received by other threads that complete the final processing of the trip.

Two synchronization mechanisms will be used in the system design: mutex and condition variables.

Mutexes

A mutex is a locking mechanism to avoid the problems that can occur when threads try to update a shared variable. We must use a mutex to ensure that only one thread at a time can access the variable. More generally, mutexes can be used to ensure atomic access to any shared resource, but protecting

shared variables is the most common use. Provides mutual exclusion, supporting ownership and other protocols. A mutex is initially created in the unlocked state in which it can be acquired by a task. After being acquired, the mutex moves to the locked state. When the task releases the mutex, it returns to the unlocked state.

The mutexes used in the main process and daemon process are listed below.

- **C_LocalSystem::mutexSoundMsg:** mutex associated with the condition variable condSoundMsg to protect the access of the global variable soundMsg;
- **C_Daemon::mutexEARclass:** mutex associated with the condition variable condEARclass to acquire a camera video;
- **C_Daemon::mutextEARshared:** protects the access and alteration of the shared memory.

Condition Variables

A condition variable allows one thread to inform other threads about changes in the state of a shared variable or other shared resource and allows the other threads to wait (block) for such notification. It allows a thread to sleep (wait) until another thread notifies (signals) it that it must do something, i.e., that some "condition" has arisen that the sleeper must now respond to.

The condition variables used in the main process and daemon process are listed below.

- **C_LocalSystem::condSoundMsg:** notifies T_Alert that a new sound message should be produced.
- **C_Daemon::condEARclass:** notifies T_CamProcess to perform a classification of the obtained set of frames.

Signals

A signal is a notification to a process that an event has occurred. Signals are sometimes described as software interrupts, being analogous to hardware interrupts in that they interrupt the normal flow of execution of a program. One process can send a signal to another process, being in that way, employed as a synchronization technique. As discussed earlier, two types of signals will

occur, since 2 different sound messages are used. The signals used in the main process and the daemon are listed below.

- **SIGUSR1:** signal sent by dameon process to the main process, whenever a state of drowsiness is detected by the camera. Specifically, this signal should be sent by the T_CamProcess thread that requires a quick response when drowsiness is detected. This is received in the main process in a signal handler, that will wake up the thread T_Alert, through the condition variable condSoundMsg, and identify which kind of message should be sent.
- **SIGUSR2:** this signal will occur after a Bluetooth connection is established between the system and another device. This way, whenever someone connects to the device, a welcome sound message will be played..

Inter-process Communication

POSIX Message queue

A POSIX message queue allow processes to exchange data in the form of messages to accomplish their tasks. They enable processes to synchronise their reads and writes to speed up processes. With this in mind, three message queue will be used for communication between the daemon process and the main process. Each thread in which its worker is processing sensor data will send the needed value for the message queue, referred to as **msgQueueSensors**. Consequently, the main process will not be responsible for to the sampling reading necessary for the camera module, will only be informed when necessary through the message queue. A message queue will also be used to receive the PID value of the LocalSystem process, since the daemon process needs to know its PID to send a signal later. This message queue will be called **msgQueuePid**. Finally, another message queue will be used to send characters via Bluetooth to the application. Since the process needs to know the mac address for sending, whenever a connection occurs in the listening thread, that address is placed in the message queue, named **msgQueueBluet**.

Thread Priorities

Threads are scheduled for execution based on their priority, where the scheduler will always pick the thread that is ready to execute with the highest

priority level, thus being important to assign each thread a static priority level to indicate their relative urgency. Priorities must be assigned to ensure efficiency and execute real-time tasks. As a result, the figures below is the priority assignment diagram for the local system's main process as well as for the daemon.

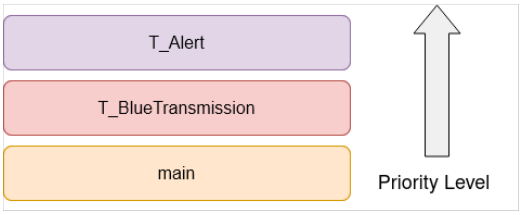


Figure 38: Priority Assignment Diagram for the Local System's Main Process

With this in mind, the priority order for each thread in the Local System main process is represented in Figure 38, and in Figure 39 for the daemon process.

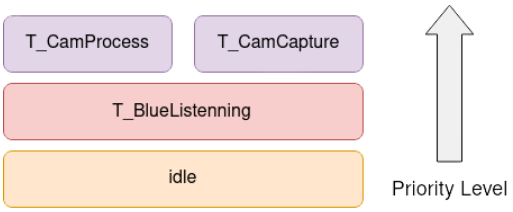


Figure 39: Priority Assignment Diagram for the Local System's Daemon Process

CPU Affinity

CPU affinity, or CPU pinning or "cache affinity", enables the binding and unbinding of a process or a thread to a central processing unit (CPU) or a range of CPUs, so that the process or thread will execute only on the designated CPU or CPUs rather than any CPU [11]. As you can see in the Figure 40 you can see an example of how CPU affinity works.

Imagine, that exists three applications: X, Y and Z. The default behavior of the scheduler is to use all the available CPUs to run the threads of applications X, Y and Z. Using the default settings, you can see that you'll get a good number of cache misses as the application is spread across all CPUs. Which leads to fewer cache hits and more cache misses.

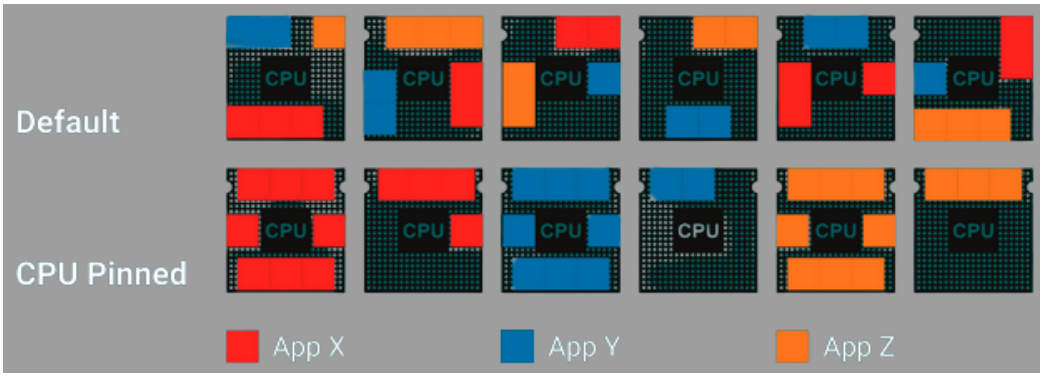


Figure 40: CPU affinity example

When the applications are pinned to specific CPUs, they're forced to run on specific CPUs thus using CPU cache more effectively. For this purpose it is possible to use the taskset command-line tool.

Usually, it's the kernel that determines the CPUs a process runs. Every time the scheduler reschedules a process, it can go to any of the available CPUs. Although this is fine for the majority of workloads, sometimes it's desirable to limit which CPU(s) a process is allowed to run.

In our project since we will be working with image processing and since our development board has a quad-core it makes sense to limit this to just one CPU. This increases the chances of more cache hits thus resulting in a much better performance.

Flowcharts

This section is intended to explain in more detail the routines and subroutines of the system in order to get a better understanding between the class diagram representation and the threaded application.

- **Main Process**

In Figure 41 is represented the main routine and the subroutines associated, init and run. Simply put, the attributes and synchronization objects are initialized, the signal period is assigned and the callbacks associated with the signal (these callbacks are going to be specified in the next flowcharts). Finally, all threads belonging to the class are created and wait for their execution to finish, using the join function.

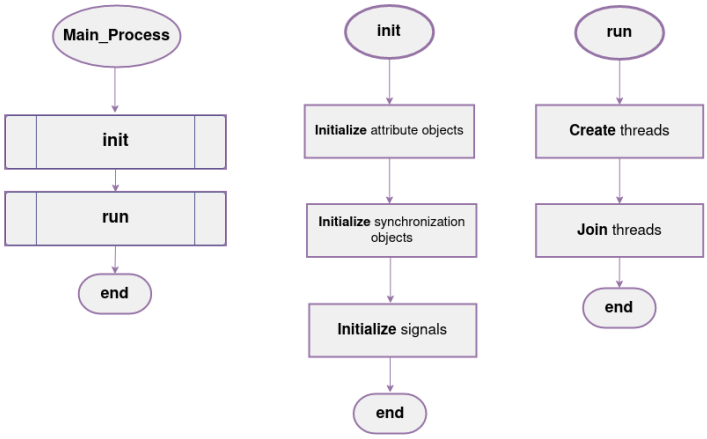


Figure 41: Main flowchart

Next, in Figure 42 is presented, BlueTransmission function, i.e. the T_BlueTransmission worker, responsible for transmitting to the remote system, via Bluetooth, the number of times the alert was activated, and other information stored during the trip.

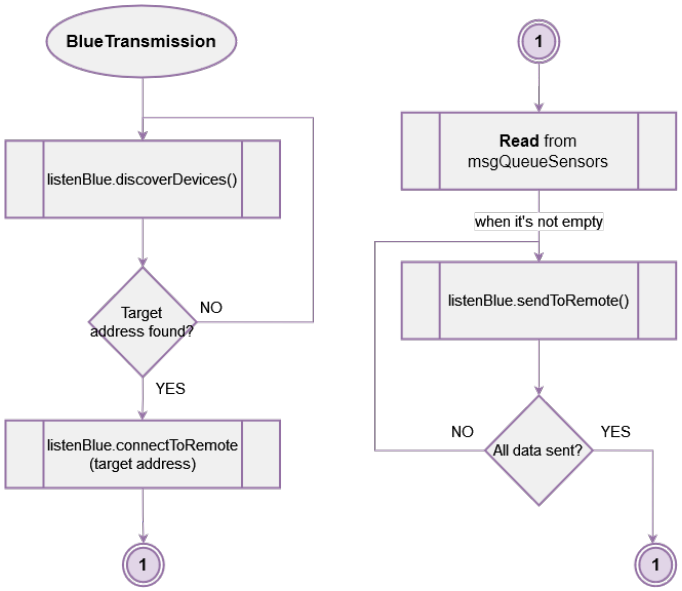


Figure 42: BlueTransmission flowchart

The data are sent to the message queue (msgQueueSensors) at the end of trip by the daemon process, and then removed from the message queue by the main process in T_BlueTransmission worker.

Then in Figure 43 is the signal handler of the main process being in charge of signaling the condition variable condSoundMsg when a SIGUSR1 signal is catched. Basically, when a signal is identified by the handler, a global variable is changed, which must be atomic and volatile. This variable takes on a specific value, depending on which message you want to play.

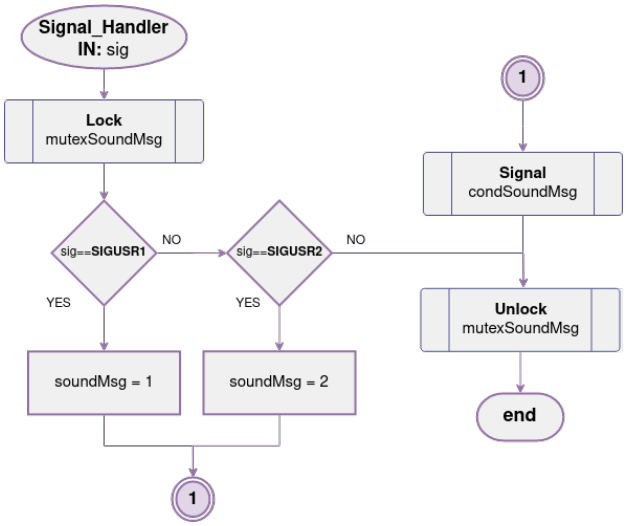


Figure 43: Signal Handler flowchart

Finally, in Figure 44 is presented, the work function of thread T_Alert. It executes and performs the necessary sound processing to alert the driver all the times it detects drowsiness. As referenced above, the global variable soundMsg is an argument to the setAlarm function, so we resorted to using mutex to provide mutual exclusion for its access. The mutex used was named mutexSoundMsg. The setAlarm function simply analyses the value of the soundMsg variable and outputs its associated sound file.

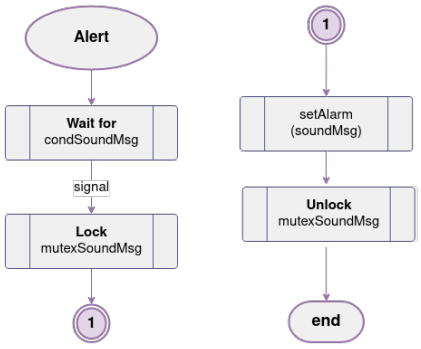


Figure 44: Alert flowchart

- **Daemon Process**

In the Figure 45 is represent the main flowchart of the daemon process. Firstly, is followed the procedure to created the daemon, then its created the message queue responsible to transfer the trip data at the end of the trip, to the main process. Finally, is create the threads needed and is executed a join, similar to the main process.

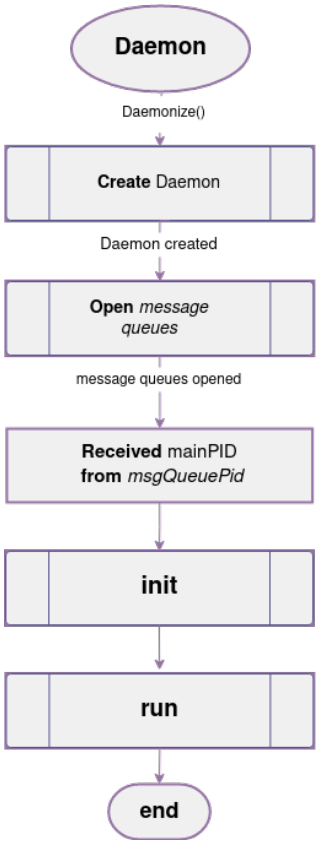


Figure 45: Daemon flowchart

Next, its going to be demonstrated one of the main process subroutine in the program, the T_CamCapture worker, Figure 46. This thread starts by the initialization of the camera and then goes to a while loop. In this loop is captured a video frame that is stored in the MAT type, that's a type from OpenCV library used to store image data. This same frame is returned to the processParameter() function, which will use algorithms provided by the OpenCV library to estimate a set of essential parameters to obtain the driver's drowsiness level. These parameters are nothing more than the position of points that make up the contours of the eyes, Figure 29.

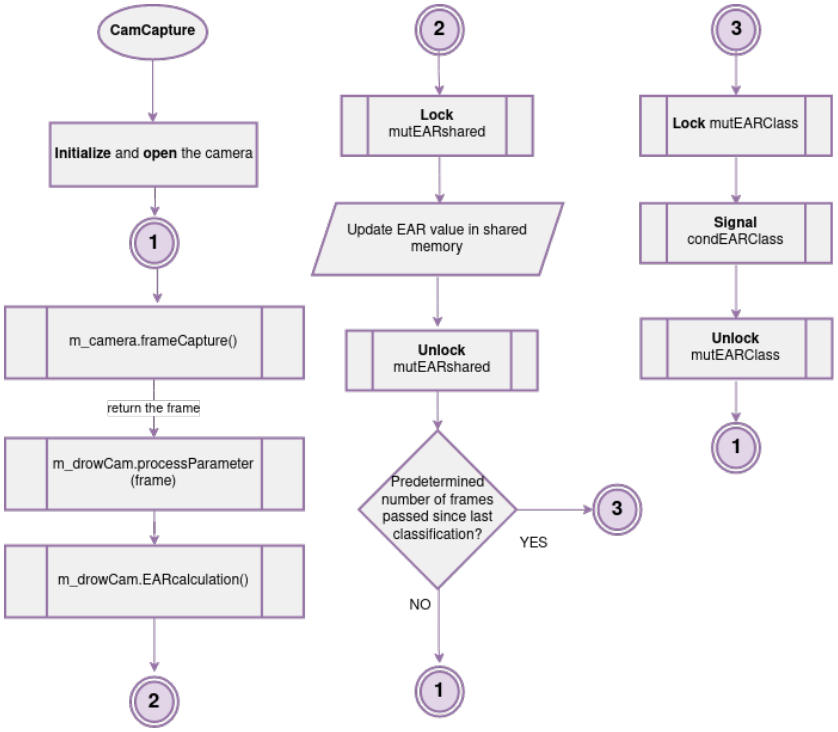


Figure 46: CamCapture flowchart

After obtaining the position of these parameters, the EAR will be calculated, as shown in the section on drowsiness detection. Next, it will check the state of the mutex associated with the shared memory for communication between the two tasks, and if possible add the new calculated value to the memory. If the stipulated number of frames for a new classification is reached, a signal will be given associated with the condition variable condModelClass, which will induce the thread worker CamProcess to proceed to a new classification of the set of frames stored in the shared memory, Figure 47.

Basically the function CamProcess waits for a signal associated with the condition variable condModelClass. When the signal is received it then proceeds to the processing responsible for determining the user's state of sleepiness - studied in the section on drowsiness detection. Briefly, this algorithm will receive an input matrix, which corresponds to the last 15 EAR values obtained, and determine whether in that time set open eye (0), short blink (1) and long blink (2) have been identified. Then it will be checked by two rules whether the previous values indicate situations in which the driver should be warned about drowsiness, rules that are also explained in the section on

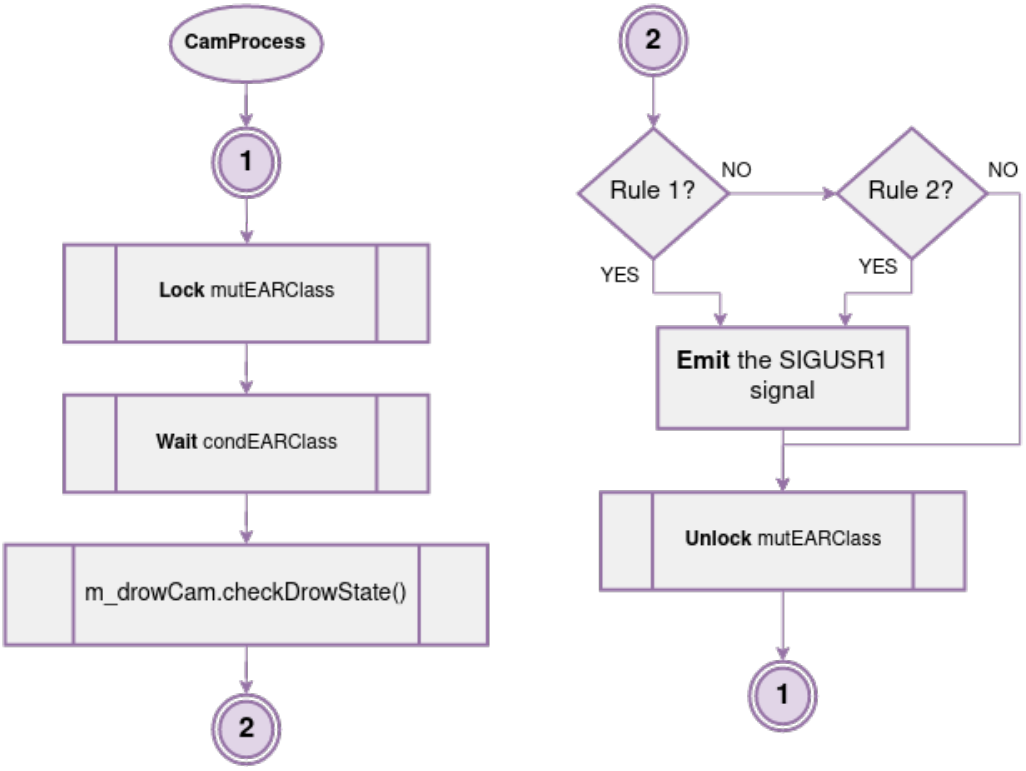


Figure 47: CamProcess flowchart

drowsiness detection. It is important to note that for signaling the main process, a user signal provided by Linux will be used.

Next, in Figure 48 is presented T_BlueListening worker, responsible for receiving the commands from the remote system, via Bluetooth, to initiate the trip or to stop the trip. If it the receives a start command to initiate a trip, enables the signal to the sensors acquisition, but when receives a stop command to finish the trip it will generate an event.

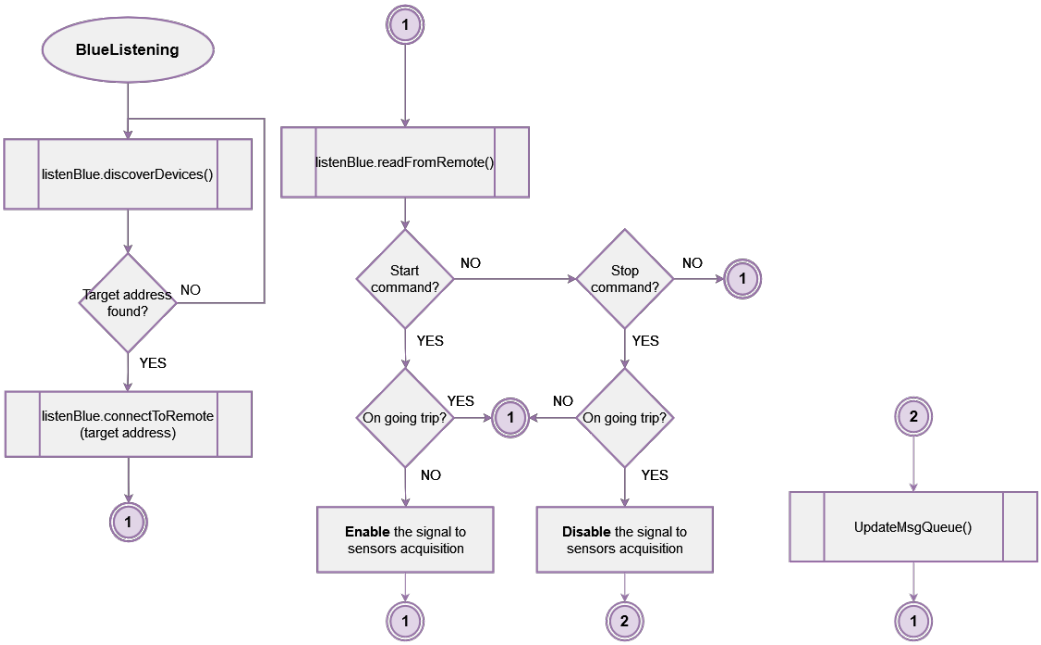


Figure 48: BlueListening flowchart

Test Cases

It is important to know how the system will behave in certain events, thus being able to predict various outputs. The purpose is to ensure that each function is performed the way it is intended so that when the entire system is together everything works as it should. In Table 6 are shown the Local Software System test cases to this module.

Test Case	**Expected Output**	**Real Output**
Bluetooh Communications		
Establish connection with the Remote System	Connection established	-
Receive commands from the Remote System	Command received	-
Process commands	Execute command if valid	-
Send processed data to the Remote System	Processed data sent	-

Drowsiness Calculation		
Calculate Drowsiness Level	Drowsiness calculated correctly	-
Speaker		
Set alarm	Plays the alarm sound on the speaker	-
Stop alarm	Alarm sound stopped	-
Device		
Light up the green LED	Green LED on	-
Camera		
Open camera device	Device opened correctly	-
Capture camera video	Variable containing video frames	-
Eyes detection	Obtain the eyes state	-
Face detection	Obtain the face state	-

Table 6: Local Software System Test Cases

5.4.4 Remote Software System

As discussed above, we will have an application that will give a GUI to the user and allow the user to choose from the various functionalities specified in the use case diagram - Figure 14. Remember, the application will make it possible to start a trip, start transmitting data to the database after a trip is finished, and exit the application.

The developed application will communicate with two subsystems: Local System via Bluetooth and with the Database server via TCP/IP protocol. When the user indicates the end of the trip, a connection must be assured between the application and the Local System, in order to proceed with the automatic transfer of the data collected during the trip. If the user wants to update the data to the database server, the database server must have access to the Wi-Fi network on the device and, of course, there must be information to send. Before the sending process is completed, the driver must log in so that the sent information is associated with his profile.

Application

Mostly the application to be developed has to interpret the commands executed by the user, perform the respective command processing, and update the user interface. So to this end, all the commands that will be interpreted by the application, and the action triggered by the command, are demonstrated below.

- **Start Command**

  When the start command is identified by the application, the system should inform the Local System that a trip has been started, and update the display to the respective trip layout.

- **Stop Command**

  When the stop command is identified by the application, the system should inform the Local System that a trip has been terminated, and update the display to the respective layout. In addition, the application should synchronize with the Local System the data collected during the trip.

- **Cancel Command**

  When the cancel command is identified by the application, the system should inform the Local System that a trip has been canceled, and update the screen to the respective layout.

- **Send to database command**

  When the update to database command is identified by the application, the system shall initiate the transmission of the stored trips to a server. First it should check if there are any trips to send, informing the user if not. Next, it should update to a display the layout for logging in. The verification and validation should be identified by the database and if they are invalid the application should alert the user of the failure. Finally, if all the previous steps are valid, the application will display a layout while the data is being sent and the sending success or failure.

GUI layouts

One of the roles of the application is to provide a graphical user interface and to enable the user to choose commands, commands specified above. The following demonstrates the various layouts that will be displayed depending

on the user's selection. It is essential that these layouts are user-friendly, and above all simple so as not to make the device difficult to use.

As soon as the user opens the application he should be faced with a menu very similar to the left image shown in Figure 49.

Figure 49: Main menu

It is shown 2 choices provided to the user: Start a trip and transfer the recorded trip data to the database server.

The diagram on the right in Figure 49 represents an idea of possible layout when the user starts a trip. In this mode, the user can terminate the trip and cancel the trip, the difference occurs at the data synchronization level, since the second option discards the synchronization with the Local System.

If the user chooses to perform a data transfer, he must log in. Figure 50 demonstrates a possible design for this process.

If the user has created an account, he only needs to fill in the fields with his email and password, as the diagram on the left shows. If the user needs to create an account, the layout on the right will be displayed. When the user login is validated, the system will proceed to send the data to the server.

In Figure 51, the diagram on the left represents a possible layout to be displayed when the user needs to select a file to collect the data from. Still, when the right layout let the user store trip data into the mobile file system, as is also shown in Figure 51.

Figure 50: Login layout

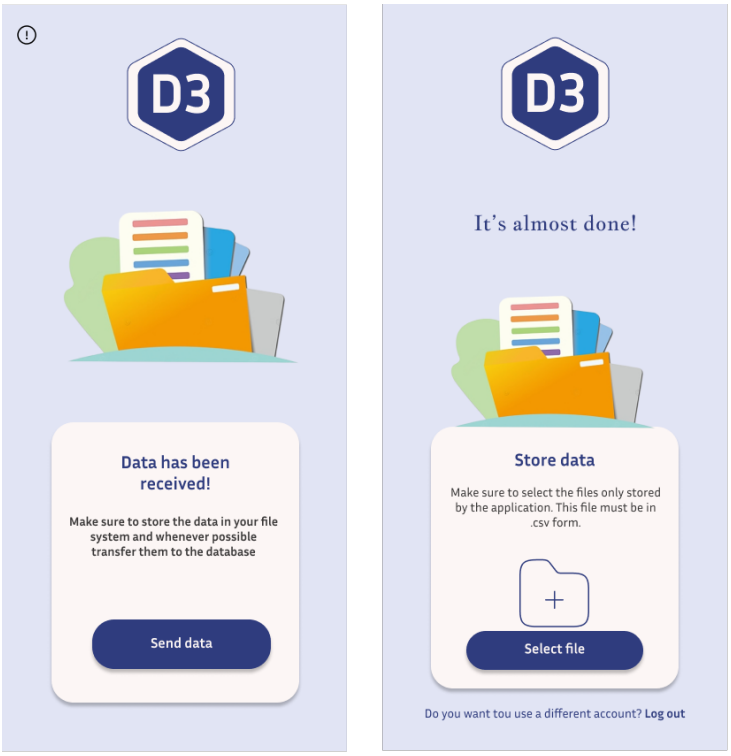


Figure 51: Transfer Layouts

Flowcharts

Next will be shown the flowcharts that represent the system's behavior. Since this is a mobile application, the developed software will interpret each touch on the screen by the user, and check if any of that touch corresponds to the selection area of any command. Figure 52 shows the system's flowchart on the whole.

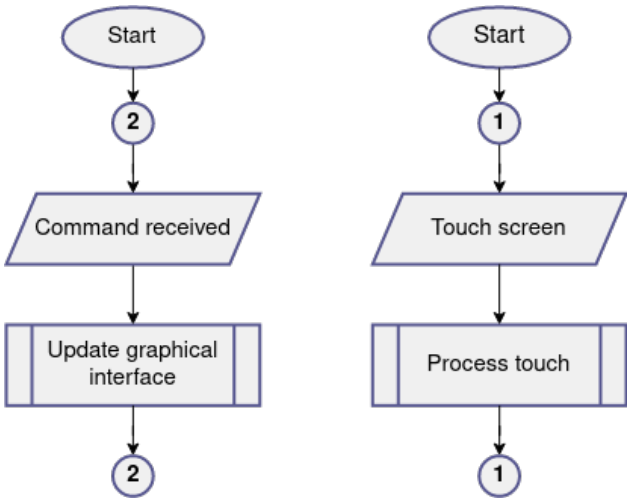


Figure 52: General flowchart

With this flowchart one can see that preemptive multitasking will occur, while one thread processes the touch screen, another thread will work on constantly updating the display. Regarding the flowchart on the left, it will receive an indication to change the layout, and to which one it should be changed. This function will use the design of the layouts previously specified for different modes in the application.

The flowchart on the right is more complex, since it deals with all the processing behind a command. In Figure 53 the Process touch subroutine is demonstrated in more detail. Considering the commands identified above, processing will be performed for each one.

If the start command is initiated, the Local system will be informed of the beginning of a trip and the display will be signaled to update the layout, as shown in Figure 54. The cancel and stop commands are very similar, it is the way the Local system interprets the commands that is different, even so when the stop command is identified, it becomes necessary to perform a data synchronization, as already mentioned.

This connection will be received through the implemented Bluetooth protocol, and if there is no error in its transmission, the user will receive a mes-

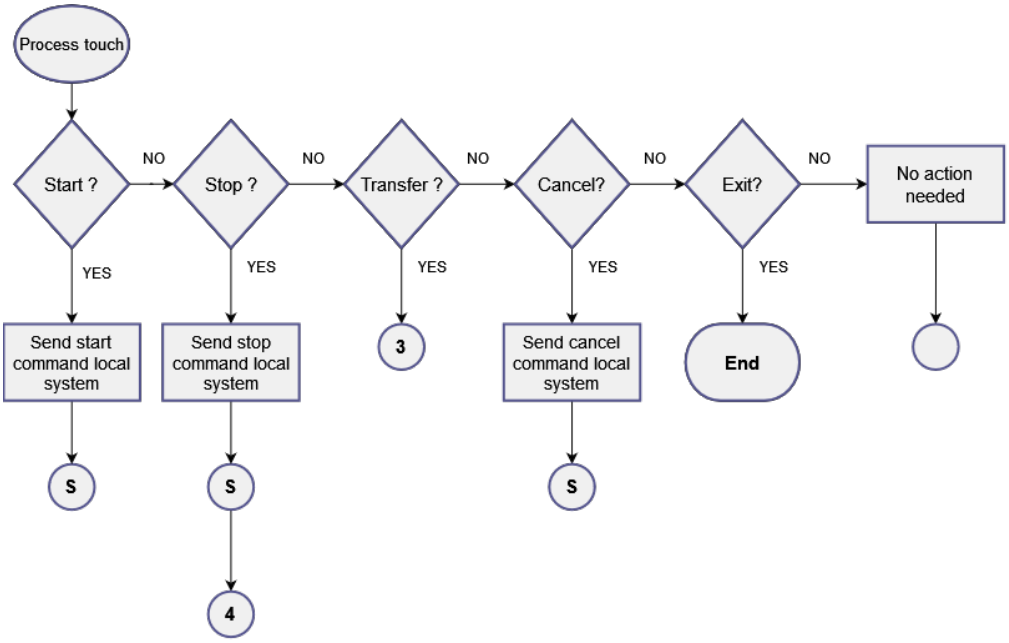


Figure 53: Process touch flowchart

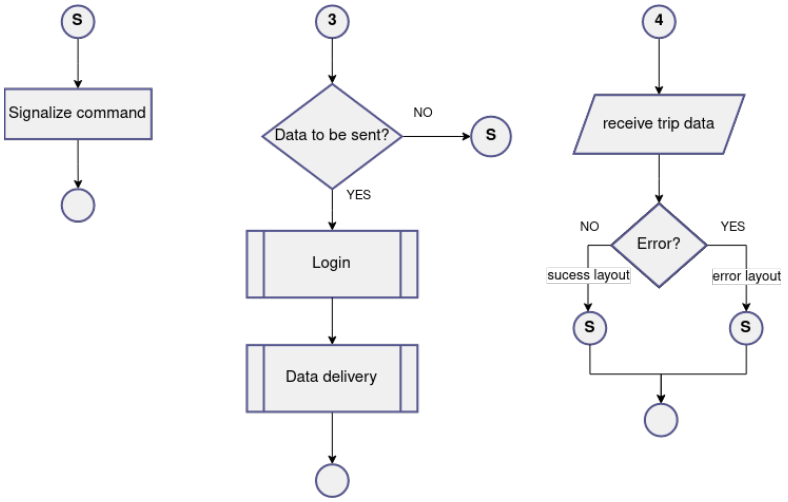


Figure 54: Additional subroutines flowchart

sage that the trip was successfully uploaded. Obviously, the exit command terminates the process, closing the application. If the user wishes to transfer the stored trip data to the database server, two subroutines, the login subroutine and the data delivery subroutine are executed, as shown in Figure 54, however, the system needs to check whether there are any trips to send.

The figure 55 shows in more detail the implementation of subroutines.

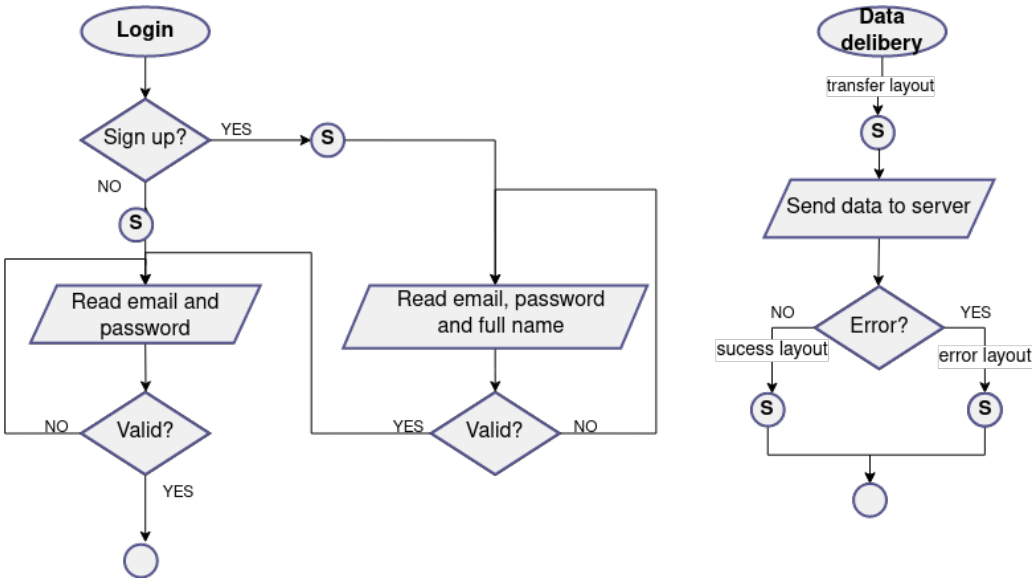


Figure 55: Data transmission flowchart

Basically, in the login subroutine, it will be identified if the user intends to sign in or needs to create an account. If the user needs to sign up, the system will read several fields and communicate with the database server to verify its validation. If they are valid, you will be redirected to the sign in layout and enter the data again. If the data entered is valid, the login process is finished. The system then proceeds to send the information associated with the user. Also in Figure 55, the layout will be updated, and will start sending the data to the database server through the TCP/IP protocol. When the transfer is finished, the user is notified with the status of the transmission, if errors have occurred he will be notified with an error layout, if no errors have occurred with a success layout. Once the transmission process is complete the user can return to the start menu and begin a new trip.

Test Cases

It is important to know how the system will behave in certain events, thus being able to predict various outputs. The purpose is to ensure that each function is performed the way it is intended so that when the entire system is together everything works as it should. In Table 7 are shown the Remote Software System test cases to this module.

Test Case	Expected Output	Real Output
Bluetooh and Wi-fi Communications		
Establish connection with the Local System	Connection established	-
Send commands to the Local System	Command received	-
Establish connection with the Server via Wi-fi	Connection established	-
Database		
Open the database	Database opened and ready to use	-
Insert/Update data into the database	Insert/Update done successfully	-
Get data from the database	Receive requested data	-
Mobile Application Communication		
Register account	Update database with new account	-
Login into account	Get the login credentials from the database and make a comparison with them	-
Logout of account	Logout valid	-

Table 7: Remote Software System Test Cases

5.4.5 Database

As mentioned in the analysis phase, the user will be able to send the trip data to a database server, where the actual data for each driver's trip will be stored. The data to be saved will always be associated with an account, which the user needs to log into before sending. Next it will be used Entity Relationship diagram [14] to define the structure of the data elements and to set a relationship between them. The database to be used will be in the form of a relational database since it will have two tables, and it can relate to each other using special keys. In Figure 56 are identified the entity types and the relationship with the cardinality: the number of possible instances participating in the relationship.

As showed in the figured, the entity types are User and Trip. The attributes of the User entity type are the name, and the email and password used in the login process. Regarding the trip entity type, the attributes are related to every data acquired during the trip, from the trip duration to the number of alerts registered. Since a trip can only have one user and a user

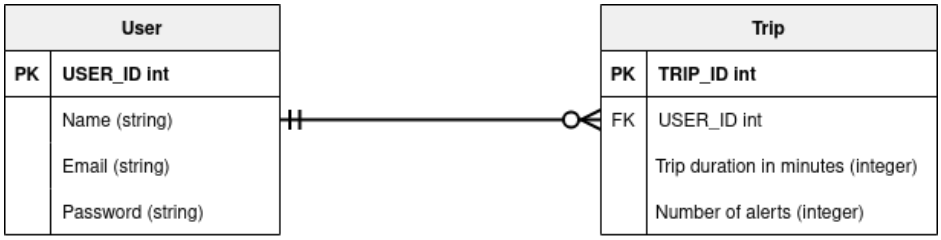


Figure 56: Entity Relationship diagram

can have multiple trip entities or even none, the relationship presented was an one to many optional.

Finally, one can see the representation of primary and foreign keys to relate the tables. For the primary key of the User entity an id was defined for each user that takes on an integer value and must be unique for each user. In the Trip entity type, there is a primary key to identify the entities and a foreign key that is nothing more than the primary key of the User entity type.

It remains to specify which methods are required between the remote system and the database server to allow the application to read and write information to the database. The methods used will be as follows:

- **Add**, if the user wants to create an account or add trip information;
- **Get**, to check if the user is registered;
- **Update**, if the user needs to change his password.

This completes the specification of all three subsystems. In the next section the dry runs performed for a given system algorithm will be presented.

5.4.6 Dry Run Tests

A dry run or practice run is a software testing process where the effects of a possible failure are intentionally mitigated. Dry run means running the program roughly on paper, in order to find out whether the program is working as it should or not. So it was decided to choose a previously implemented algorithm and perform the test. The algorithm chosen was the drowsiness determination algorithm depicted in Figure 28.

Line Number	Capture Frame	EAR	Classification	Open Eye	Short Blink	Long Blink	Rule 2	Drowsiness
1	0	-	-	-	-	-	-	0
2	1	0.30	0	-	-	-	-	0
3	1	0.29	0	-	-	-	-	0
4	1	0.27	0	-	-	-	-	0
5	1	0.25	0	-	-	-	-	0
6	1	0.20	1	-	-	-	-	0
7	1	0.19	0	-	-	-	-	0
8	1	0.20	0	-	-	-	-	0
9	1	0.20	0	-	-	-	-	0
10	1	0.22	0	-	-	-	-	0
11	1	0.24	1	-	-	-	-	0
12	1	0.23	0	-	-	-	-	0
13	1	0.23	0	-	-	-	-	0
14	1	0.26	0	-	-	-	-	0
15	1	0.27	0	-	-	-	-	0
16	1	0.27	1	0	1	0	-	0
17	1	0.27	0	-	-	-	-	0
18	1	0.27	0	-	-	-	-	0
19	1	0.28	0	-	-	-	-	0
20	1	0.29	0	-	-	-	-	0
21	1	0.28	1	0	0	0	-	0
22	1	0.29	0	-	-	-	-	0
23	1	0.30	0	-	-	-	-	0
24	1	0.29	0	-	-	-	-	0
25	1	0.29	0	-	-	-	-	0
26	1	0.28	1	0	0	0	-	0
27	1	0.27	0	-	-	-	-	0
28	1	0.26	0	-	-	-	-	0
29	1	0.26	0	-	-	-	-	0
30	1	0.26	0	-	-	-	-	0
31	1	0,26	1	1	0	0	.	0

Table 8: Open Eye and Eye Blink Dry Run for Detection Based on 1 Rules

Line Number	Capture Frame	Open Eye	Short Blink	Long Blink	Time	Rule 1	Rule 2	Drowsiness
1	0	0	0	0	0	-	0	0
2	1	1	0	0	5	-	0	0
3	1	1	0	0	10	-	0	0
4	1	1	0	0	15	-	0	0
5	1	0	1	0	20	-	0	0
6	1	0	1	0	25	-	0	0
7	1	0	1	0	30	-	0	0
8	1	0	1	1	35	-	0	0
9	1	0	0	0	40	-	0	0
10	1	1	0	0	45	-	0	0
11	1	1	0	0	50	-	0	0
12	1	0	1	0	55	-	0	0
13	1	0	1	0	60	0.16	0	0
14	1	0	1	0	65	0.14	0	0
15	1	1	0	0	70	0.14	0	0
16	1	1	0	0	75	0.14	0	0
17	1	0	1	0	80	0.13	0	0
18	1	1	0	0	85	0.14	0	0
19	1	0	1	0	90	0.14	0	0
20	1	0	1	1	95	0.14	0	0
21	1	0	0	0	100	0.25	0	1
22	1	1	0	0	105	0.16	0	0
23	1	0	1	0	110	0.13	0	0
24	1	0	0	1	115	0.22	0	1
25	1	0	0	1	120	0.33	0	1
26	1	0	0	1	125	0.44	0	1
27	1	0	0	1	130	0.56	0	1
28	1	0	0	1	135	0.67	1	1

Table 9: Open Eye and Eye Blink Dry Run for Detection Based on 2 Rules

6 Implementation

In this chapter will be discussed the D3 implementation. All system setup steps will be described and explained to bring the project to realization.

6.1 System Configurations

In this section the various steps for completing the settings of our system will be presented. First, the process for setting up the Buildroot will be demonstrated, then the necessary settings for the OpenCV, Qt Creator IDE and finally the settings for our database creation tool, MySQL.

6.1.1 Buildroot

Regarding the Buildroot, the tool used to generate the project's embedded Linux system through cross-compilation, the installation is simply done by executing the follow steps.

```
$ cd ~
$ mkdir buildroot
$ cd buildroot
$ wget https://buildroot.org/downloads/buildroot-2021.08.tar.
   gz
$ tar xzf buildroot-2021.08.tar.gz
$ cd buildroot-2021.08
```

Once the installation is successfully completed, there are several makes for configuring the toolchain, however, only the ones below will be used.

```
$ make raspberrypi4_defconfig
$ make menuconfig
$ make
$ make clean
```

The first command is needed to configure the kernel image for the Raspberry Pi 4, as it performs the necessary settings regarding hardware usage and collection of some board packages. Subsequently the the two next commands give to the programmer a graphical interface, Figure 57, to set up the toolchain.

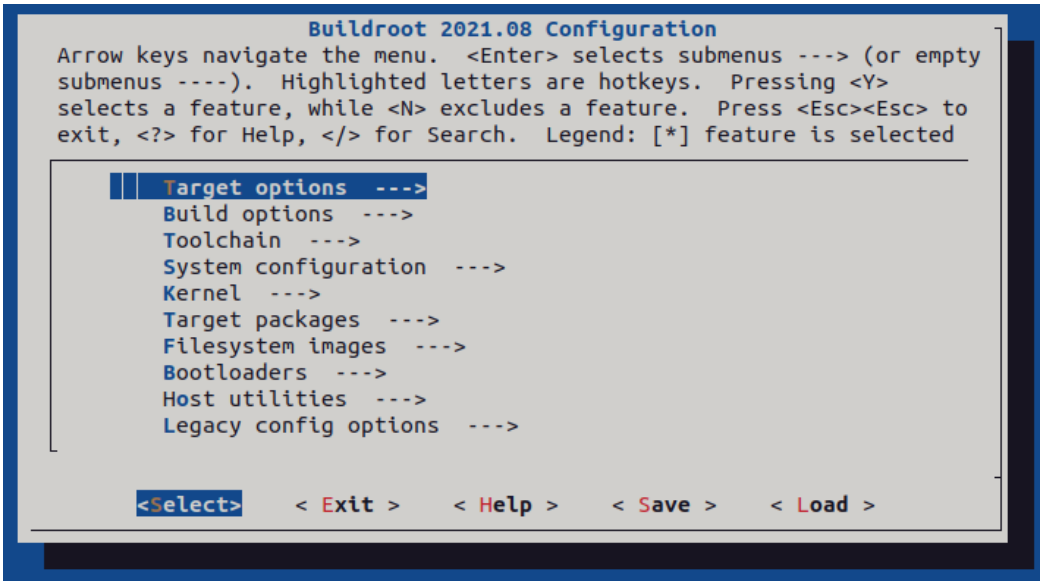


Figure 57: Buildroot menuconfig.

Once in the Buildroot using **make menuconfig** one can use a graphic interface to select packages and change configurations in order to create a custom linux image.

Under **System configuration** select:

- **dev management**,select:
  - `Dynamic usign devtmpfs + eudev`: To ensure that all new devices connected to the Raspberry Pi generate an event that can be recognized. Devtmpfs is a virtual filesystem within the Linux kernel, when mounted on /dev, this virtual filesystem will automatically make device files appear and disappear as hardware devices are added and removed from the system. The eudev is a daemon that runs in the background, and is called by the kernel when a device is added or removed from the system;

Under **Target-Packages** select:

- **Audio and Video Applications**, select:
  - `alsa-utils`
    - `alsactl`: command-line utility that is part of the ALSA, used to manage and manipulate the settings of ALSA sound;

        * `alsamixer`: is a command-line mixer utility for ALSA;
        * `aplay/arecord`: aplay and arecord are command-line utilities for playing and recording audio on Linux systems;
        * `speaker-test`: a command-line utility in Linux that is used to test audio output on a system;
    - `ffmpeg`: for video and audio processing;
    - `gstreamer 1.x`: library for constructing graphs of media-handling components;
    - `v4l2grab`: utility for grabbing JPEG images;
    - `v4l2loopback`: allows you to create virtual video devices;

- **Hardware Handling**, select:
    - `eudev`: to have access to libudev;
    - `rpi-userland`: Used to detect camera hardware;
    - `dbus`: the D-Bus message bus system;
    - `Hardware Handling`
        * `Firmware`
            · `brcmfmac-sdio-firmware-rpi/brcmfmac-sdio-firmware-rpi-bt`: Bluetooth module firmware;
            · `rpi-firmware/rpi 4 (extended)`: the extended set of files for versions 4;

- **Intrepreter languages and scripting**, select:
    - `tcl`
        * `Tcl libraries/modules`
            · `expect`: extension to the Tcl scripting language that provides a simple way to automate interactions with other programs;

Libraries, select:

- `Audio/Sound`
    - `alsa-lib`: library that provides the ALSA interface to applications;

- `Hardware Handling`

  - `bcm2835`: C library for Raspberry Pi user programs;
  - `libv4l`: select `v4l-utils tools`: API for real time video capture;

- `Text and terminal handling`
  - `ncurses/enable wide char support`: enable wide char and UTF-8 support;

- `Graphics`
  - `OpenCV3`: select `jpeg support`, `png support`, `v4L support`, `videoio`, `highgui`, `imgcodecs`, `imgproc`, `v4l support`: Used for image capturing and processing;

- `Multimedia`
  - `libcamera`: a complex camera support library;
  - `x264`: for encoding video streams into the H. 264/MPEG-4 AVC compression format;

Networking applications, select:

- `dhcpcd`: open source DHCP client;
- `dropbear`: small SSH server which will let us log in remotely;
- `bluez-utils and all the secondary packages`: provides stack, library and tooling for Bluetooth Classic and Bluetooth Low Energy;
- `bluez-tools`: A set of tools to manage Bluetooth devices for linux;

BusyBox, select:

- `Linux Sytem Utilities`
  - `taskset` a command-line utility in Linux that is used to set or retrieve the CPU affinity of a process or task. Used to control which CPUs a process or task is allowed to run on, which can be useful for image processing.

6.1.2 OpenCV

Having set up OpenCV in Buildroot, to make the compiled OpenCV libraries available on Raspbery Pi, some configuration is needed to make cross-compiling with OpenCV possible. The procedure for installing the OpenCV libraries is shown below.

```
$ cd ~/buildroot/buildroot-2021.08/output/build/opencv3
   -3.4.15/buildroot-build
$ cmake-gui
$ cmake .
$ make
$ sudo make install
```

An important note is that it is assumed that cmake and cmake-gui are already installed, otherwise they must be installed first. After being in the above directory, using the first command, when using cmake-gui you should click on "configure" and activate all the necessary OpenCV packages (imgproc, videoio, etc). At the end click "Generate" and exit. The following commands are straightforward. These libraries are installed in the usr/lib directory, making it possible to cross-compile with OpenCv.

6.1.3 MySQL

To successfully manage the MySQL database service, you first need to install it correctly. Below are the steps for the respective installation.

```
$ sudo apt upgrade
$ sudo apt-get install libmysqlclient-dev
$ sudo apt install mysql-server
```

When the installation is finished, it is necessary to create a strong password and become a superuser, then its possible to enter into mysql by the folowing command:

```
$ mysql -u root -p
```

With all these steps completed, it should be noted that the application will be deployed to an android device, and so **Android Studio** was used.

6.2 Image Generation

Finalized all the necessary tools, packages and configurations, its possible to create the Linux custom image, that will run on the Raspberry Pi.

```
$ cd ~/buildroot/buildroot-2021.08/
$ make
```

Completed the make, you can copy the image to the SD card, which will go into the Raspberry Pi. You can use the Linux dd command, you must define the input file, if, which is the Linux image, and the output file, of, which is the SD card.

```
$ sudo dd if=./output/images/sdcard.img of=/dev/sda
```

6.3 System Initialization

After generating the image it is still necessary to make some changes to the boot file, the **config.txt** file. The listing 1 shows the necessary configuration. For this project we added the statement in the 22 line in order to enable the mini UART Bluetooth, and the statement in line 24 that enables the audio output of Raspberry Pi 4.

```
# We always use the same names, the real used variant is selected by
# BR2_PACKAGE_RPI_FIRMWARE_{DEFAULT,X,CD} choice
start_file=start.elf
fixup_file=fixup.dat

kernel=zImage

# To use an external initramfs file
# initramfs rootfs.cpio.gz

# Disable overscan assuming the display supports displaying the full resolution
# If the text shown on the screen disappears off the edge, comment this out
# disable_overscan=1

# How much memory in MB to assign to the GPU on Pi models having
# 256, 512 or 1024 MB total memory
gpu_mem_256=256
gpu_mem_512=256
gpu_mem_1024=256

# fixes rpi (3B, 3B+, 3A+, 4B and Zero W) ttyAMA0 serial console
dtoverlay=miniuart-bt

# Audio
dtparam=audio=on

# Overclock
```

```
28 arm_freq=2000
29 gpu_freq=750
30 over_voltage=6
```

Listing 1: The /boot/config.txt file.

6.3.1 Init Scripts

Init scripts, also known as "system initialization" scripts, are scripts that are run during the boot process of a Linux-based operating system. They are usually located in the /etc/init.d/ directory, and are used to start and stop various system services and daemons.

The naming convention for init scripts typically follows the format of "[SK]nn<init.d filename>", where:

- **SK**: is a prefix indicating that the script is a system initialization script, where S means start this job and K means kill this job.
- **nn**: is a two-digit number that determines the order in which the script is run during the boot process. The lower the number, the earlier the script is run.
- <init.d filename>": is the name of the script, which typically corresponds to the name of the service or daemon that the script controls.

In order to ensure that the programs developed are executed after the system boots, its necessary to create the respective init scripts, following the procedure detailed in Listing 2.

```
1 $ cd /etc/init.d
2 $ vi S<scriptName>.sh
3 $ chmod +x S<scriptName>.sh
```

Listing 2: Procedure for creating an init script.

The following two init scripts for the local system have been implemented and will be added to the /etc/init.d directory, how its possible to see in Listing 3 and 4. This first script is written in a shell language, bash, for setting up and running some Bluetooth commands for connection on a Linux system. Initially is loaded the *hci_uart* kernel module, which provides support for Bluetooth communication over a serial port, the *hciattach /dev/ttyS0 bcm43xx 921600 noflow* command for attaching a Bluetooth device to the serial port at */dev/ttyS0* using the *bcm43xx* driver with a baud rate of 921600 and the noflow option. The line *sdptool add SP* uses the sdptool command to add a Service Provider (SP) entry to the Bluetooth Service Discovery

Protocol (SDP) database. In a row the line */etc/init.d/blue.sh >/dev/null 2>1* runs the blue.sh script, for bluettoth connection, in the background and redirects the standard output and standard error streams to */dev/null*, effectively suppressing any output from the script. For the messages queues there is the need to create a directory called */dev/mqueue* to store message queue files and to mount the message queue file system type (mqueue) at the directory */dev/mqueue* with *mount -t mqueue none /dev/mqueue*.

The last lines runs the dSensors.elf program as a daemon process and d3.elf program as the main process.

```
#!/bin/sh

echo "Inserting bluetooth-tools..."
modprobe hci_uart
hciattach /dev/ttyS0 bcm43xx 921600 noflow
sdptool add SP

/etc/init.d/blue.sh >/dev/null 2>&1 &

mkdir /dev/mqueue
mount -t mqueue none /dev/mqueue

# Daemon process
/root/dSensor.elf

# Main process
/root/d3.elf
```

Listing 3: Init script for local system.

This second script is written in Expect, the scripting language that can be used to automate interactions with other programs, especially over a network. After starting the bluetoothctl program, the command line interface for Bluetooth control on Linux systems, there was a need to wait for some commands and depending on the expect send the correct command.

Initially its spawned bluetoothctl starting the interface for Bluetooth control on Linux systems. Afterwards it waits for several expects from bluetoothctl and the respective sends are sent. Finally when requested a confirmation automatically sends a *yes* response, Listing 4.

```
#!/usr/bin/expect

spawn bluetoothctl
expect "Agent registered"
send "power on\r"
expect "Changing power on succeeded"
send "discoverable on\r"
expect "Changing discoverable on succeeded"
send "pairable on\r"
expect "Changing pairable on succeeded"
send "default-agent\r"
expect "Default agent request successful"

while {1} {
    expect {
        "Request confirmation" {
        send "yes\r"
        }
        eof {
            break
        }
    }
}
```

Listing 4: Bluetooth script.

6.4 CMakeLists

The *CMakeLists.txt* is a file used by the cross-platform, open-source build system *CMake* for managing the build process of our project. The file lists the source files, library dependencies, and build options of the project and defines the steps that *CMake* will use to build the project. The file contains instructions for compiling the source code into the executable file (.elf), generating project files for other build tools, and setting properties such as the minimum required version of *CMake* or the location of dependencies.

With this type of file, using *CMake* it is possible to generate build files for a specific build system, such as Makefiles.

In the particular case of our project, D3, two *CMakeLists.txt* were developed, one for the Daemon and another for the Main process, thus generating two different makefiles.

The first file is the Daemon CMakeLists, in the Listing below 5. Its required the OpenCV library and the use of Arm G++ and GCC compilers. The script sets the path to the OpenCV library, to the path to the Arm compilers, and sets the directory where the output executable file will be stored.

Going through the file is defined two things. The directories for including header files in the project, including the directories for OpenCV, the current source directory *inc*, and the parent source directory. All source files used in the project.

Then is added an executable file named *dSensor.elf* to the project. The executable file is built from several source files, including files for daemon *(CDaemon.cpp)*, device camera *(CDevCamera.cpp)*, drowsiness detection *(CDrowsinessCam.cpp)*, bluetooth communication *(CBluetoothCom.cpp)* and acquisition *(dAcquisition.cpp)*. Finishing its linked the OpenCV library, thread library, and Bluetooth library to the executable file.

```
project( CLocalSystem LANGUAGES CXX)
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED on)

#OpenCV Package
find_package( OpenCV REQUIRED )

# Compilers
set(CMAKE_CXX_COMPILER "/home/andre/buildroot-2021.08/output/
    host/bin/arm-buildroot-linux-uclibcgnueabihf-g++")
set(CMAKE_CC_COMPILER "/home/andre/buildroot-2021.08/output/
    host/bin/arm-buildroot-linux-uclibcgnueabihf-gcc")

# Executable directory
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY "${
    CMAKE_CURRENT_SOURCE_DIR}")

# OpenCv include DIRs
set(OpenCV_INCLUDE_DIRS "/usr/include/opencv2")
set(OpenCV_INCLUDE_DIRS "/usr/lib")

# Including all includes DIRs
include_directories( include ${OpenCV_INCLUDE_DIRS} )
include_directories( include ${CMAKE_CURRENT_SOURCE_DIR}/inc/
    )
include_directories( include ${CMAKE_CURRENT_SOURCE_DIR}/../
    inc )

# All Source files
file(GLOB all_SRCS
    "${CMAKE_CURRENT_SOURCE_DIR}/src/*.cpp"
    "${CMAKE_CURRENT_SOURCE_DIR}/src/*.c"
    "${CMAKE_CURRENT_SOURCE_DIR}/../src/*.cpp"
)

# Daemon process
add_executable( dSensor.elf ${all_SRCS} )
```

```
33
34 target_link_libraries( dSensor.elf ${OpenCV_LIBS} ${
       CMAKE_THREAD_LIBS_INIT} ${CMAKE_CXX_FLAGS} -lbluetooth)
```

Listing 5: Daemon CMakeLists.

The second file is the Main process CMakeLists, in the Listing below 6. The procedure was practically the same, differing only in one aspect. In this file there is no need to include the OpenCV libraries.

Is added an executable file named *d3.elf* to the project. The executable file is built from several source files, including files from the local main (localmain.cpp), for local system *(CLocalSystem.cpp)*, bluetooth communication *(CBluetoothCom.cpp)* and sound *(CDevSound.cpp)*. Finishing its linked the thread library, and Bluetooth library to the executable file.

```
1  project( CLocalSystem LANGUAGES CXX)
2  set(CMAKE_CXX_STANDARD 11)
3  set(CMAKE_CXX_STANDARD_REQUIRED on)
4
5  # Compilers
6  set(CMAKE_CXX_COMPILER "/home/andre/buildroot-2021.08/output/
       host/bin/arm-buildroot-linux-uclibcgnueabihf-g++")
7  set(CMAKE_CC_COMPILER "/home/andre/buildroot-2021.08/output/
       host/bin/arm-buildroot-linux-uclibcgnueabihf-gcc")
8
9  # Executable directory
10 set(CMAKE_RUNTIME_OUTPUT_DIRECTORY "${
       CMAKE_CURRENT_SOURCE_DIR}")
11
12 # Including all includes DIRs
13 include_directories( include ${CMAKE_CURRENT_SOURCE_DIR}/inc/
       )
14 include_directories( include ${CMAKE_CURRENT_SOURCE_DIR}/../
       inc )
15
16 # All Source files
17 file(GLOB all_SRCS
18    "${CMAKE_CURRENT_SOURCE_DIR}/src/*.cpp"
19    "${CMAKE_CURRENT_SOURCE_DIR}/src/*.c"
20    "${CMAKE_CURRENT_SOURCE_DIR}/../src/*.cpp"
21 )
22
23 # Main process
24 add_executable( d3.elf localmain.cpp CLocalSystem.cpp
       CBluetoothCom.cpp CDevSound.cpp ${all_SRCS} )
25 target_link_libraries( d3.elf ${CMAKE_THREAD_LIBS_INIT} ${
       CMAKE_CXX_FLAGS} -lbluetooth)
```

Listing 6: Main process CMakeLists.

6.5 Makefiles

As described before, the Makefiles for the Daemon process and Main process were generated by using CMake via the CMakeLists files. Thus, there is no need to go deep down in this particular Makefiles. On the other hand one that should be mentioned is the GPIO Led character device driver Makefile. This can be observed in the Listing 7. It allows generating the file *led.ko* that is used to mount the device driver.

```
obj-m := led.o
led-objs := ledmodule.o utils.o

KDIR := /home/andre/buildroot-2021.08/output/build/linux-custom/
ARCH ?= arm
CROSS_COMPILE ?= /home/andre/buildroot-2021.08/output/host/bin/arm-buildroot-linux-uclibcgnueabihf-

export ARCH CROSS_COMPILE

KBUILD_TARGETS  := clean help modules modules_install

.PHONY: all $(KBUILD_TARGETS)

all: modules
$(KBUILD_TARGETS):
  $(MAKE) -C $(KDIR) M=$(shell pwd) $@
```

Listing 7: Makefile Device Driver.

6.6 Device Driver

6.6.1 Led Device Driver

In order to turn on a led whenever the user starts a trip, a GPIO device driver has been developed. The device driver will be a simple character mode device driver. To set the pins as output and set their state, high or low) an auxiliary file, utils, was used as like the one in the Listing 8.

```
#include "utils.h"
#include <linux/module.h>

void SetGPIOFunction(struct GpioRegisters *s_pGpioRegisters, int GPIO, int functionCode) {

int registerIndex = GPIO / 10;
int bit = (GPIO % 10) * 3;

```

```
unsigned oldValue = s_pGpioRegisters->GPFSEL[registerIndex];
unsigned mask = 0b111 << bit;

s_pGpioRegisters->GPFSEL[registerIndex] = (oldValue & ~mask)
    | ((functionCode << bit) & mask);
}
void SetGPIOOutputValue(struct GpioRegisters *
    s_pGpioRegisters, int GPIO, bool outputValue) {

     if (outputValue){
         s_pGpioRegisters->GPSET[GPIO / 32] = (1 << (GPIO %
    32));
     }
     else{
         s_pGpioRegisters->GPCLR[GPIO / 32] = (1 << (GPIO %
    32));
     }

}
```

Listing 8: Functions to SetGPIO and SetGPIOOutputValue.

The device driver will have six functions (initiate, exit, open, close, read and write). The last one writes logic values into a pin so that can turn on a led, using the function's on the Listing 8. Henceforth, the composition of the functions that make up this device driver are presented in detail.

The file structure, Listing 9, represents an open file. It is created by the kernel on open and it is passed to any function that operates on the file, until the last close. Note that the **read** function is defined regardless of not being used, since it is a write-only device.

```
//File operation structure
static struct file_operations ledDevice_fops = {
  .owner = THIS_MODULE,
  .write = led_device_write,
  .read = led_device_read,
  .release = led_device_close,
  .open = led_device_open,
};
```

Listing 9: Led Device Driver implementation (File Operations Structure).

To begin with, is performed the opening of the device driver.

```
//This function will be called when we open the Device file
int led_device_open(struct inode* p_inode, struct file *
    p_file){

     /* Print a msg with the function being called */
     pr_alert("%s: called\n",__FUNCTION__);
```

```
6
7       /* Store the handle to the GPIO registers */
8       p_file->private_data = (struct GpioRegisters *)
        s_pGpioRegisters;
9       return 0;
10
11  }
```

Listing 10: Led Device Driver implementation (Open Function).

The close function, Listing 11, only closes the led device driver.

```
1  // This function will be called when we close the Device file
2  int led_device_close(struct inode *p_inode, struct file *
       pfile) {
3
4       /* Release the handle to the GPIO registers */
5       pfile->private_data = NULL;
6       return 0;
7  }
```

Listing 11: Led Device Driver implementation (Close Function).

Depending on the entry in pbuff, the write function, Listing 12, allows the setting of whether the pin's value is high or low.

```
1  // This function will be called when we write the Device file
2  ssize_t led_device_write(struct file *pfile, const char
       __user *pbuff, size_t len, loff_t *off) {
3
4       struct GpioRegisters *pdev;
5
6       /* Check if we have a valid handle */
7
8       if(unlikely(pfile->private_data == NULL))
9           return -EFAULT;
10
11      /* Get GPIO handle */
12      pdev = (struct GpioRegisters *)pfile->private_data;
13
14      if (pbuff[0]=='0') /**< If user space issued 0 */
15          SetGPIOOutputValue(pdev, LedGpioPin, 0); /**< Turn
        off led */
16      else
17          SetGPIOOutputValue(pdev, LedGpioPin, 1); /**< Turn on
        led */
18      return len;
19 }
```

Listing 12: Led Device Driver implementation (Write Function).

The initialization is accomplished by the function in Listing 13. It allocates a range of character device numbers. The main number will be chosen dynamically, and returned. In addition, it creates, initializes, a character device and map bus memory for CPU space.

```
1  // Module Init function
2  static int __init led_device_init(void) {
3
4    int ret;
5    struct device *dev_ret;
6
7    /* Allocatesa range of char device numbers. The major
      number will be chosen dynamically, and returned */
8      if(( ret = alloc_chrdev_region(&ledDevice_majorminor, 0,
      1, DEVICE_NAME)) <0){
9      return ret;
10     }
11
12   /* IS_ERR is used to check if class create succeded
      creating ledDevice_class . If an error occurs unregister
13     â   the char driver and signal the error . */
14
15   if(IS_ERR(ledDevice_class = class_create(THIS_MODULE,
      CLASS_NAME))){
16     unregister_chrdev_region(ledDevice_majorminor, 1);
17     return PTR_ERR(ledDevice_class);
18   }
19
20   /* Creates a device and registers it */
21   if(IS_ERR(dev_ret = device_create(ledDevice_class, NULL,
      ledDevice_majorminor, NULL, DEVICE_NAME))){
22     class_destroy(ledDevice_class);
23     unregister_chrdev_region(ledDevice_majorminor, 1);
24     return PTR_ERR(ledDevice_class);
25   }
26
27   /* Initialize device */
28     cdev_init(&c_dev, &ledDevice_fops); /**< Map device
      functions */
29     c_dev.owner = THIS_MODULE;
30
31     /* Try to add a char device to the system */
32     if ((ret = cdev_add(&c_dev, ledDevice_majorminor, 1)) <
      0) {
33
34          printk(KERN_NOTICE "Error %d adding device", ret);
35          device_destroy(ledDevice_class, ledDevice_majorminor)
      ;
36          class_destroy(ledDevice_class);
```

```
37          unregister_chrdev_region(ledDevice_majorminor, 1);
38          return ret;
39      }
40
41      /* Get GPIO handle remapped to virtual address */
42      s_pGpioRegisters = (struct GpioRegisters *)ioremap(
        GPIO_BASE, sizeof(struct GpioRegisters));
43
44      /* Print the virtual address */
45      pr_alert("map to virtual address: 0x%x\n", (unsigned)
        s_pGpioRegisters);
46
47      /* Set the GPIO as output */
48      SetGPIOFunction(s_pGpioRegisters, LedGpioPin, 0b001); //
        Output
49
50    return 0;
51  }
```

Listing 13: Led Device Driver implementation (Init Function).

Completing, the elimination of the device driver is performed by the function exit, Listing 14. It configures the pin as input and erases the device registers.

```
1  // Module exit function
2  static void __exit led_device_exit(void)
3  {
4    SetGPIOFunction(s_pGpioRegisters, LedGpioPin, 0); //
       Configure the pin as input
5    iounmap(s_pGpioRegisters);
6    cdev_del(&c_dev);
7    device_destroy(ledDevice_class, ledDevice_majorminor);
8    class_destroy(ledDevice_class);
9    unregister_chrdev_region(ledDevice_majorminor, 1);
10 }
```

Listing 14: Led Device Driver implementation (Exit Function).

The functions presented in Listing 15 are used to insert and remove the device driver, through the insmod and rmmod commands, respectively.

```
1  module_init(led_device_init);
2  module_exit(led_device_exit);
```

Listing 15: Led Device Driver implementation (Insert and Remove functions).

6.7 Main Process

In this section we want to demonstrate the implementation of the main process code. To do this, segments of the code will be presented and explained.

Following will then be a description of all the modules used.

6.7.1 Bluetooth Class

For the implementation of the bluetooth class, and following the same procedure for the other modules, we implemented initialization functions, demonstrated in listing 16.

```
1  bool CBluetoothCom::init(char *destAux,int channel)
2  {
3      strcpy(this->m_dest,destAux);
4      str2ba(this->m_dest, &m_addr.rc_bdaddr);
5      m_addr.rc_channel = (uint8_t) channel;
6
7      return true;
8  }
9
10 bool CBluetoothCom::init(int channel)
11 {
12     m_addr.rc_channel = (uint8_t) channel;
13
14     return true;
15 }
```

Listing 16: Bluetooth class initialization.

It was decided to implement function overloading at initialization, since there can be two types of inits to establish Bluetooth communication. The first function (line 1 to 8) is used if you have a mac address to connect to, while the second function (line 10 to 15), initializes the *CBluetoothCom* class object as a Bluetooth server, i.e., it will wait for connection requests on the respective channel specified by *m_addr.rc_channel*.

The implemented Bluetooth class has two pairs of functions that act depending on the object you want to create. It is important to note that each object created can only be used either to send characters or to receive characters.If the communication that is intended for the object is that it assumes the role of Bluetooth transmitter, i.e., an object that will establish a connection with another device and send characters, the pair of functions to interact with the other device is given by the listing 17.

```
1  int CBluetoothCom::connectToRemote()
2  {
3      return connect(this->m_socket, (struct sockaddr *)&this->
       m_addr, sizeof(this->m_addr));
4  }
5
6  int CBluetoothCom::sendToRemote(char* buf, int size)
```

```
7 {
8     return write(this->m_socket, buf, size);
9 }
```

Listing 17: Bluetooth class transmission.

The first function *connectToRemote* allows you to create a Bluetooth socket, providing a communication channel for data exchange between applications running on different devices. If the connection is valid, you can then use the *sendToRemote* function, which outputs the character set stored in the *buf* variable to the other device.

However, for receiving characters from other devices the function pair to use is represented in the listing 18

```
1  bool CBluetoothCom::listenRemote(char *targetAddr)
2  {
3      struct sockaddr_rc rem_addr = { 0 };
4
5      bind(this->m_socket, (struct sockaddr *)&this->m_addr,
       sizeof(this->m_addr));
6
7      listen(this->m_socket, 1);
8
9      this->m_client = accept(this->m_socket, (struct sockaddr
       *)&rem_addr, &this->m_opt);
10
11     this->m_addr = rem_addr;
12     ba2str( &this->m_addr.rc_bdaddr, targetAddr);
13     return true;
14 }
15
16 int CBluetoothCom::sendToRemote(char* buf, int size)
17 {
18     return write(this->m_socket, buf, size);
19 }
```

Listing 18: Bluetooth class listening.

These functions differ from the previous one in that, instead of the object requesting a connection to another device, the *listenRemote* function expects a connection request from another device. When a character set is received, it is stored in *buf*, as the line 18 shows.

6.7.2 Sound Device Class

In the listing 19, the function responsible for audio reproduction is represented. As can be seen, two different voice messages can be reproduced.

```
1 int CDevSound::setAlarm(int SoundMsg) const
```

```
2 {
3
4     cout << SoundMsg << endl;
5     switch (SoundMsg)
6     {
7     case 1:
8         system("timeout 3 aplay -q /root/DetectDrowsiness.wav
      ");
9         break;
10
11    case 2:
12        system("timeout 4 aplay -q /root/Welcome.wav");
13        break;
14    default:
15        cerr << "Error value!" << endl;
16        break;
17    }
18    return 1;
19 }
```

Listing 19: CDevSound class.

6.7.3 Local System Class

Regarding the *CLocalSystem* class, this presents greater complexity. Not the entire implementation of the class will be presented, however you can always consult the remaining implementation in the link provided in the bibliography.

First the necessary initializations are shown, listing 20. As specified earlier, the PID of the main process, line 5, is obtained and its value is sent to the message queue *msgQueuePid*. This step is crucial, since the daemon process needs to know the PID of this process to send an alarm signal later.

```
1  void CLocalSystem::init()
2  {
3      int error;
4      char pid[5];
5      int pid_aux = getpid();
6
7      signal(SIGUSR1, signal_Handler);
8      signal(SIGUSR2, signal_Handler);
9      signal(SIGINT, signal_Handler);
10
11     sprintf(pid, "%d", pid_aux);
12     error = mq_send(this->msgQueuePid, pid, 6, 1);
13     if(error == -1)
14     {
15         error = errno;
```

```
16          if(error != EAGAIN)
17              std::cerr << "In mq_send()";
18              exit(1);
19      }
20
21      pthread_mutex_init(&mutexSoundMsg, NULL);
22      pthread_cond_init(&condSoundMsg, NULL);
23
24      pthread_attr_init (&thread_attr);
25    pthread_attr_getschedpolicy (&thread_attr, &thread_policy);
26    pthread_attr_getschedparam (&thread_attr, &thread_param)M
27
28      SetupThread(25,&thread_attr,&thread_param);
29      pthread_create(&T_BluetTransmission_id, &thread_attr,
     BluetTransmission, this);
30
31      SetupThread(25,&thread_attr,&thread_param);
32      pthread_create(&T_Alert_id, &thread_attr, Alert, this);
33  }
```

Listing 20: Bluetooth class listening.

In this listing, the creation and prioritization of threads is also observed. The priority value selected was the one that the system presented the highest performance overall.

Next, the behavior adopted by each thread belonging to the class will be demonstrated.

Starting with the BLU thread, it waits in blocking for the MAC address of the device that has connected to the system to be updated in the message queue, as show in listing 21. After receiving the address, the Bluetooth connection is initialized to send data to the remote device. However, this sending will only occur when data is received in the message queue *msgQueueSensors*, line 6. Next, the system uses the member functions of the *CBluetoothCom* class to connect to the device and send the data.

```
1  error = mq_receive(ptr->msgQueueBluet, targetAddr, 10000,
     NULL);
2
3  while (1)
4  {
5      error = mq_receive(ptr->msgQueueSensors, dataFromSensors,
       10000, NULL);
6
7      status = ptr->m_sendBlue.connectToRemote();
8
9      if (status == 0)
10     {
11         char dataSend[20];
```

```
12          sprintf(dataSend, "%s,%d,", dataFromSensors, ptr->
    alertNumber-1);
13          status = ptr->m_sendBlue.sendToRemote(dataSend, 20);
14          ptr->alertNumber = 0;
15      }
16  }
```

Listing 21: Code segment of T_BluetTransmission worker.

As you can see in the 13 line, the data sent to the remote device is separated by a comma, and the number of alerts is subtracted by one. This is because when sending the Linux *SIGUSR2* signal, the number of alerts is also incremented. As this signal is only transmitted once during program execution, the total number of alerts is subtracted by one.

A code segment of the *CAlert* thread is visible in the listing 22.

```
1 pthread_mutex_lock(&ptr->mutexSoundMsg);
2
3 pthread_cond_wait(&ptr->condSoundMsg, &ptr->mutexSoundMsg);
4 ptr->alertNumber++;
5 ptr->m_speaker.setAlarm(ptr->soundMsg);
6
7 pthread_mutex_unlock(&ptr->mutexSoundMsg);
```

Listing 22: Code segment of T_Alert worker.

Thus, whenever a conditional signal associated with the mutex *mutexSoundMsg* and the conditional variable *condSoundMsg* is sent, the system executes the function *setAlarm*, which takes as argument the type of sound message you want to play. If it is an alert it will take value one, if it is a welcome message it will take value two. As mentioned above, note that whenever a sound message is required, the number of alerts is increased.

6.8 Daemon Process

Now will be explained some code excerpts implemented in the *CDaemon* class for the correct functioning of the system. The remaining classes will be covered later in the related **Drowsiness Detection and Warnings** chapter.

6.8.1 Daemon Class

Following the procedure of the Main process, the listing 23 demonstrates the initialization needed for the implementation of Class *CDaemon*.

```
1 pthread_mutex_init(&mutexEARclass, NULL);
2 pthread_mutex_init(&mutexEARshared, NULL);
3
```

```
4  pthread_cond_init(&condEARclass, NULL);
5
6  pthread_attr_init (&thread_attr);
7  pthread_attr_getschedpolicy (&thread_attr, &thread_policy);
8  pthread_attr_getschedparam (&thread_attr, &thread_param);
9
10 SetupThread(25,&thread_attr,&thread_param);
11 pthread_create(&T_BluetListening_id, &thread_attr,
       BluetListening, this);
12
13 SetupThread(25,&thread_attr,&thread_param);
14 pthread_create(&T_CamCapture_id, &thread_attr, CamCapture,
       this);
15 SetupThread(25,&thread_attr,&thread_param);
16 pthread_create(&T_CamProcess_id, &thread_attr, CamProcess,
       this);
17
18 m_listenBlue.init();
```

Listing 23: Code segment of CDaemon initialization.

As seen above, mutexes and condition variables are initialized. Subsequently, the methods used to configure the priorities of each thread are observed. The selected priority was the one that verified the highest system performance.

Regarding the worker thread, some excerpts are represented in the Listing 24 and Listing 25, will now be explained in detail.

```
1 error = mq_receive(ptr->msgQueuePid, msg, 10000, NULL);
2 ptr->localPid = stoi(msg);
```

Listing 24: Code segment one of T_CamProcess worker.

In the code above, the thread waits to receive data in the line 1. Since the program needs to know the PID of the process to which the Linux signal is to be sent, it was resorted to using the message queue *msgQueuePid*, to receive this value from the main process. This value is then converted to an integer (line 2).

Still in the same thread, a conditional signal is awaited, which identifies when a new classification of the SVM model needs to be performed, as shown in Listing 25. For this, in line 5, the function *checkDrowState* , which sends the last 15 EAR values obtained as an argument.

```
1 pthread_mutex_lock(&ptr->mutexEARshared);
2
3 pthread_cond_wait(&ptr->condEARclass, &ptr->mutexEARshared);
4 alarmValue = ptr->m_drowCam.checkDrowState(ptr->classInput);
5 pthread_mutex_unlock(&ptr->mutexEARshared);
6 if(alarmValue)
```

```
7 {
8     kill(ptr->localPid,SIGUSR1);
9 }
```

Listing 25: Code segment two of T_CamProcess worker.

This function will return a boolean, which if it has the value true, a signal will be sent to the main process, indicating that it is necessary to carry out an alert.

Regarding the thread *T_CamCapture*, the listing 26 presents part of the implemented code.

```
1  tempFrame = ptr->m_camera.frameCapture();
2  ptr->m_drowCam.processParameter(tempFrame);
3
4  pthread_mutex_lock(&ptr->mutexEARshared);
5      for(int i = MAX_FEAT - 1; i > 0; i--)
6      {
7          ptr->classInput[i] = ptr->classInput[i-1];
8      }
9      ptr->classInput[0] = ptr->m_drowCam.EARcalculation();
10
11 pthread_mutex_unlock(&ptr->mutexEARshared);
12
13 if((ptr->nextClass--) == 0)
14 {
15     pthread_cond_signal(&ptr->condEARclass);
16
17     ptr->nextClass = NUM_NEXT_CLASS;
18 }
```

Listing 26: Code segment two of T_CamCapture worker.

First, a frame is captured and the processing of this same frame is carried out, thus obtaining the necessary values for the calculation of the EAR value. After, the line 4 to 9, is an algorithm implemented that allows to shift all elements of the array *classInput* to the right. This array is the input of the SVM model and represents the 15 stored EAR elements. The last element of the array is deleted, and the new EAR value is added to the first position, line 9. Subsequently, it is checked whether the number of captured frames allows a new classification. As explained above, this decision prevents the same blink from being classified twice. If a classification can be carried out, condition variable *condEARclass* is signaled.

To finish the class *CDaemon*, it still remains to explain the worker thread *T_BluetListening*.

In listing 27, Bluetooth communication is initialized with a selected fixed channel. The thread will be blocked until a connection is accepted with

another device. When this connection occurs, the *SIGUSR2* signal is sent to the main process, which will issue a welcome message.

```
ptr->m_listenBlue.init(READ_CHANNEL);
ptr->m_listenBlue.listenRemote(targetAddr);
kill(ptr->localPid,SIGUSR2);
```

Listing 27: Code segment one of T_BluetListening worker.

The second excerpt that appears in the Listing 28, intends to better explain how the received Bluetooth messages are interpreted.

```
bytes_read = ptr->m_listenBlue.readFromRemote(buf,sizeof(buf));
    if(bytes_read > 0)
    {
        switch(buf[0])
        {
            case 'S':
                error = mq_send(ptr->msgQueueBluet, targetAddr, sizeof(targetAddr)/sizeof(targetAddr[0])+1, 1);

                ptr->tripStatus = true;
                write(ptr->file_descriptor , &ptr->LedOn, 1);
                start = std::chrono::steady_clock::now();

                break;
            case 'T':
                end = std::chrono::steady_clock::now();
                time_dif = std::chrono::duration_cast<std::chrono::minutes>(end - start).count();

                sprintf(dataSample,"%d",time_dif);

                error = mq_send(ptr->msgQueueSensors, dataSample, 4, 1);
                write(ptr->file_descriptor , &ptr->LedOff, 1);
                ptr->tripStatus = false;
                break;
            case 'C':
                write(ptr->file_descriptor , &ptr->LedOff, 1);
                ptr->tripStatus = false;
                break;
            default:
                cerr << "[recvBlut] Invalid command receiveid" << endl;
        }
```

```
31
32      }
```

Listing 28: Code segment two of T_BluetListening worker.

In this code, whenever characters are received via Bluetooth, the first character of each message is analyzed and thus, it is possible to identify which command is to be executed. Thus, the command received by Bluetooth will be:

- **S** : signals the start of the trip, turns on the LED that indicates the trip status (line 10), and sets a time reference point to later calculate the trip time (line1 1).
- **T** : signals the end of the trip, turns off the LED that indicates the trip status (line 21), and calculates the trip time (line 16). This value is then sent to the *msgQueueSensors*.
- **C** : signals trip cancellation, turns off the LED that indicates the trip status (line 25), and ignores the timer trip values.

The implementation of the code belonging to the classes is now finished, and now the remaining algorithms of the system will be explained.

6.9 Eye Landmarks Detection

Essentially, in order to obtain the eye landmarks, one needs to capture a frame using the Raspberry Pi camera, as shown in Listing 29.

As formerly stated before, the eye landmarks detection algorithm is based on OpenCV.

```
1  if (!camDev.isOpened())
2      ("Camera did not open!\n");
3
4
5  camDev.read(frame);
6
7
8      if(frame.empty())
9          cerr << ("Camera no frames captured.\n");
10
11 //Resizes the frame, for a better performance
12 resize(frame,frame,Size(),1.0/4,1.0/4);
13
14 return frame;
```

Listing 29: Capture and Returns frame from Camera.

Other important necessary implementation when capturing the frame, to get a clear image from the camera, is to calibrate the camera parameters using the code present in Listing 30 . This way is used Video for Linux (V4L) API.

```
/******** Camera Configurations **********/
camDev.set(CAP_PROP_FRAME_WIDTH , FRAME_W); //640
camDev.set(CAP_PROP_FRAME_HEIGHT, FRAME_H); //480
camDev.set(CAP_PROP_BRIGHTNESS, CAM_BRIGHTNESS); //0.5
camDev.set(CAP_PROP_CONTRAST, CAM_CONTRAST); //0.5
camDev.set(CAP_PROP_SATURATION, CAM_SATURATION); //0.5
camDev.set(CAP_PROP_FPS, CAM_FPS); //30

system("v4l2-ctl --set-ctrl=auto_exposure=0");
system("v4l2-ctl --set-ctrl=iso_sensitivity=4");
system("v4l2-ctl --set-ctrl=auto_exposure_bias=18");
system("v4l2-ctl --set-ctrl=exposure_time_absolute=1000");
system("v4l2-ctl --set-ctrl=exposure_dynamic_framerate=1");
system("v4l2-ctl --set-ctrl=white_balance_auto_preset=7");
system("v4l2-ctl --set-ctrl=sharpness=30");
```

Listing 30: Capture and Returns frame from Camera.

OpenCV's facial landmark API is called Facemark. It has three different implementations of landmark detection based on three different papers, FacemarkKazemi, FacemarkAAM, FacemarkLBF. Even though the Facemark API consists of three different implementations, a trained model is available for only FacemarkLBF [16]. Thereby the Facial Landmark Detection trained model used was lbfmodel.yaml. Its created an instance of the Facemark class. It is wrapped inside OpenCV smart pointer and loaded the landmark detector, lbfmodel.yaml in line 2. This landmark detector was trained on a few thousand images of facial images and corresponding landmarks. Public datasets of facial images with annotated landmarks can be found [17].

```
facemark = FacemarkLBF::create();
facemark->loadModel("/usr/share/OpenCV/lbfmodel.yaml");
```

Listing 31: Load landmark detector.

Another important aspect, before jumping to the code, is the face detector. All facial landmark detection algorithms take as input a cropped facial image. Therefore, the first step is to detect all faces in the image, and pass those face rectangles to the landmark detector. For this was used the OpenCV's HAAR face detector "haarcascade_frontalface_alt2.xml". Haar Cascade classifier is trained on a large set of positive and negative images, and the resulting classifier consists of a set of simple rectangular features

(Haar features) that are combined in a cascaded manner to make the final decision. The "haarcascade_frontalface_alt2.xml" file contains the parameters of these Haar features, as well as other configuration settings, that are used by the classifier to perform face detection.

```
1 CascadeClassifier faceDetector("/usr/share/OpenCV/
     haarcascades/haarcascade_frontalface_alt2.xml");
```

Listing 32: Load Face Detector.

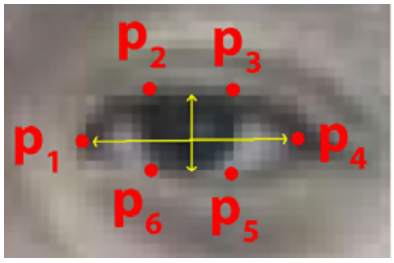
With the pre-trained models loaded, the frame was processed to obtain the EAR values. The first thing to do is to check if the frame is empty to avoid future problems. Then, since the face detector requires a grayscale image, the frame is converted to a gray frame and the face detector is rushed on each frame of the video. The output of the face detector is a vector of rectangles containing one or more faces in the image. Thereafter, is past the original image and the detected face rectangles to the facial landmark detector in line 13 of Listing 33. For every face, is obtained 68 landmarks that are stored in vector of vector of points. We will only focus on the landmarks of the eyes.

```
1  if (frame.empty())
2    cerr  << ("[Camera] blank frame captured.\n");
3
4  vector<Rect> faces;
5
6  cvtColor(frame, gray, COLOR_BGR2GRAY);
7
8  faceDetector.detectMultiScale(gray, faces);
9
10 vector<vector<Point2f>> landmarks;
11 bool success = facemark->fit(frame,faces,landmarks);
12
13 if(success)
14 {
15     for(int i = 0; i < landmarks.size(); i++)
16     {
17         this->renderFace(frame, landmarks[i]);
18     }
19 }
```

Listing 33: Processing the frame.

Once we have obtained the landmarks, its possible with the *renderFace* function to obtain every single eye landmarks. As already mentioned before, our landmarks of interest are the ones in the Figure below, 58.

Consequently it was developed the code as the Listing 34 below, in order to get each of the points necessary to obtain the result of the EAR equation.



$$\text{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$

Figure 58: Landmarks of interest.

Since all points are coordinates, it is necessary to use both x-axes and y-axes. The final value of the points subtraction are the module of the same subtraction, in order to be used in the respective equation.

```
//p2-p6
double p2_p6_x = landmarks[37].x  - landmarks[41].x;
double p2_p6_y = landmarks[37].y - landmarks[41].y;

this->p2_p6_final = sqrt((p2_p6_x * p2_p6_x) + (p2_p6_y *
    p2_p6_y));

//p3-p5
double p3_p5_x = landmarks[38].x - landmarks[40].x;
double p3_p5_y = landmarks[38].y - landmarks[40].y;

this->p3_p5_final = sqrt((p3_p5_x * p3_p5_x) + (p3_p5_y *
    p3_p5_y));

//p1-p4
double p1_p4_x = landmarks[36].x - landmarks[39].x;
double p1_p4_y = landmarks[36].y - landmarks[39].y;

this->p1_p4_final=sqrt((p1_p4_x * p1_p4_x) + (p1_p4_y *
    p1_p4_y));
```

Listing 34: Eye Landmarks.

Having obtained all the necessary elements, it is already possible to calculate the EAR value as it is present in the Listing 35.

```
double EAR = (p2_p6_final +  p3_p5_final)/(2* p1_p4_final);
return (float)EAR;
```

Listing 35: Ear Value Calculation.

6.10 SVM

Since a single threshold is not adequate for every user because each person has his own 'natural EAR' due to his facial features it was not possible to set a universal threshold, being necessary calibration for every user. Also, this approach leads to many false positive alerts, once natural expressions such as talking or smiling tended to decrease the EAR. Thus, the approach considered was to get the EARs of 15 consecutive frames, 15 features, that were concatenated to create a user state characteristic. The labels could be classified into three categories: eye open, blink, or closed eye (0,1 and 2, respectively). Specifically, the state label is only considered a blink when the touch of eyelids occurs on the six central frames of the state feature (i.e., from the 5th to the 10th frame). This procedure was adopted to avoid counting the same blink twice or more when running the detection model. Therefore, the model uses input from fifteen dimensions (15 consecutive EARs) and returns a scalar (state label).

Support Vector Machine (SVM) was used as the classification learning algorithm, once it presents the advantage that previous knowledge about neither the function behavior nor the relation between input and output is required. The problem could be seen as follows: there exists a mapping function $y = f(x)$, unknown, of real values and, possibly, non-linear between an input vector x and an output scalar y and the only available information is the data set D, called training data. This mapping will separate each region in the classification problem, i.e., delimit whether an input x (state feature) is open eye, blink, or closed eye. As aforementioned, SVM needs a set of training data to infer about state labels. Thus, a CSV file was created containing 503 label state samples. From the total of 503 samples, 371 contained open eyes, 48 contained blinking EARs, and 84 contained closed eyes EARs. This file contained data from each state's label (i.e. 0, 1, 2) and includes different gender users, natural and abnormal facial expressions, glass-wearing users, and distinct head positions. Data were collected from several users, once EAR usually differs for each person. This was done to teach the algorithm using ordinary situations when all these cases could be possible. Therefore, SVM was trained and the file mentioned before was used as a training set.

In the Figure 60 below you can see 3 examples of the CSV file used, open eye, blink and closed eye, (i.e. 0, 1, 2) respectively.

Having finalized the CSV file, the training dataset, the code below was developed. First of all, it is necessary to read the CSV file and extract all the features and their classification label, 15 EARs and their label classification line by line, as can you see in Listing 36.

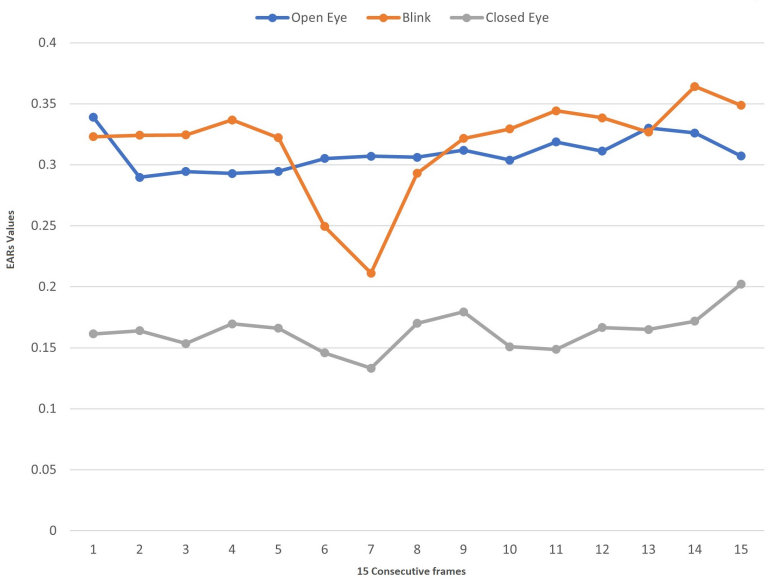


Figure 59: CSV file used as Training Set, with 3 classifications.

```
//Read the CSV file into a matrix and a vector.
ifstream file("/root/test.csv");

vector <vector<double>> data;
vector <double> labels;
string line;
string label;   //classification
string feature; //ear values

while(getline(file,line,'#'))
{
        getline(file,label,'\n');

        stringstream inputLine(line);
        vector <double> row;

        double value, label2;

        while(getline(inputLine,feature,','))
        {
            value = stod(feature);
            row.push_back(value);
        }
        data.push_back(row);
        row.clear();
        label2=stod(label);
```

```
27          labels.push_back(label2);
28 }
```

Listing 36: Read the CSV file into a matrix and a vector

Since it already has the features and labels values, its possible to create the SVM and train it and get the trained SVM model to a file, svm.xml, Listing 40. This file will later be used in the main code, being capable of making predictions from an input of 15 EARs.

Primarily its created an instance of the SVM class with **SVM::create** and and are defined some parameters before training the SVM, such as the kernel type and the values for any relevant kernel parameters [15]. The type of SVM chosen here was type **SVM::C_SVC** that can be used for n-class classification (n $\geq$ 2). The important feature of this type is that it deals with imperfect separation of classes (i.e. when the training data is non-linearly separable). Allows imperfect separation of classes with penalty multiplier C for outliers. The type of SVM kernel chosen was **SVM::RBF**, Radial basis function, since its Non-Linear Training Data. It is a mapping done to the training data to improve its resemblance to a linearly separable set of data. This mapping consists of increasing the dimensionality of the data and is done efficiently using a kernel function. Intuitively, the **Gamma** parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. Its related to the localization and flexibility for the RBF kernel. The **C** parameter trades off correct classification of training examples against maximization of the decision function's margin. For larger values of C, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. A lower C will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy. In other words C behaves as a regularization parameter in the SVM. The value chosen for this parameter was the one that gave the best accuracy.

```
1 Ptr<SVM> svm = SVM::create();
2
3 svm->setType(SVM::C_SVC);
4 svm->setKernel(SVM::RBF);
5 svm->setGamma(0.4);
6 svm->setC(10);
```

Listing 37: Set up SVM's parameters.

The function **SVM::train** that will be used afterwards requires the training data to be stored as **Mat** objects of floats. Therefore, we create these

objects like the ones below:

```
Mat trainingData(data.size(), 15, CV_32FC1);
Mat trainingLabels(labels.size(), 1, CV_32SC1);
```

Listing 38: Set up the training data.

Next it is created the training dataset for the SVM, machine learning algorithm. The data and labels arrays are input data and target labels, respectively. The code is populating a 2D matrix "trainingData" with the input data and a 1D matrix "trainingLabels" with the target labels. For each iteration of the outer loop, 0 to data.size(), the inner loop , 0 to 15, is used to populate the 15 columns of a single row in the "trainingData" matrix with the corresponding data[i][j]. Finally, the target label for this data is stored in the "trainingLabels" matrix. The outer loop will run "data.size()" times, thus creating a complete training dataset to be used in the SVM algorithm.

```
for(int i=0; i<data.size(); i++) {
    for(int j=0; j<15; j++) {
        trainingData.at<float>(i,j) = data[i][j];
    }
    trainingLabels.at<int>(i,0) = labels[i];
}
```

Listing 39: Regions classified by the SVM.

Is called the function **TrainData::create**, used to create a training dataset from the "trainingData" and "trainingLabels" matrices. The input to "TrainData::create" includes the "trainingData" matrix, a flag indicating that the data is organized in rows (ROW_SAMPLE), and the "trainingLabels" matrix.

Finally, the **trainAuto** function is called on the svm object and passed the trainData as an argument. The function trains the SVM model using the provided training dataset and the model parameters are automatically selected based on the data. With that being done the trained SVM model is saved to a .xml file which will be used later on, making it possible to make predicts.

```
Ptr<TrainData> trainData = TrainData::create(trainingData,
    ROW_SAMPLE, trainingLabels);
svm->trainAuto(trainData);

//Save the trained SVM model to a file:
svm->save("/root/svm.xml");
}
```

Listing 40: Training the SVM.

6.11 Drowsiness Detection and Warnings

Once the SVM model has been completed and it is obtained the *svm.xml* file, the time has come to make the predictions to check whether the driver is drowsy or not. After every 5 new frames received from the video frames feed, a new state label is created using the EARs from the latest set of 15 frames available, and SVM classifies this feature in one of the three classes (open eyes, short blink, long blink).

The interesting here is dealing with drowsiness detection. Thus, SVM outputs (i.e. sequence of 0, 1 and 2) are used to determine whether the user is drowsiness or not. Two different situations were considered as drowsiness states [18]:

- **1:** If within a period of 60 seconds, the proportion between the **2** output (long blinks) and the sum of **1** and **2** outputs (total number of blinks) is higher than 25% (i.e. $(2/(2+1)) \geq 0.25$);
- **2:** If 4 or more consecutives outputs (i.e. predictions of SVM) are **2** (long blinks). This occurs when the user's eyes remain closed for more than a second.

In practice, the first situation deals with many long blinks in a short lapse of time. Consecutive long blinks are highly correlated with first stages of drowsiness. The second situation deals with continuously closed eyes, which means, the user keeps the eyes closed for more than one second. This situation could be seen as a later drowsiness stage, where the attention to the task is much smaller.

This way it was developed two functions, *checkDrowState* and *getAlert-Number* to check the rules mentioned above and to count the number of times the driver was alerted, respectively.

The first function is the *checkDrowState*, as can you see in Listings 41, 42, and 43. Fundamentally is used the OpenCV library to implement a real-time alert system based on eye blinking. The first line of code creates a *1xMAX_FEAT* (number of features) matrix named *input_x*, with the data type *CV_32F*, and the input data stored in the *input* variable. Basically all captured EAR values are continuously being converted to a variable that OpenCV understands.

The next line calls the *predict* method of the SVM classifier stored in the *svm* object, using the *input_x* matrix as input, 15 values of EARs, and stores the prediction result in the *output_y* integer variable, where this output is open eyes, shor blink or long blink. (0, 1 and 2 respectively).

```
Mat input_x = Mat(1, MAX_FEAT, CV_32F, input);
int output_y = svm->predict(input_x);
```

Listing 41: Drowsiness Detection Rules.

The following lines implement a set of rules to determine if an alert should be generated based on the prediction result, Listing 42. A boolean variable is initialized to false, *alert_var*, which is specifically for when there is a drowsiness and should alert is true otherwise not. Next, the code keeps track of the number of short and long blinks, as well as the number of consecutive long blinks, and updates these values based on the prediction result. Essentially the code implements a rolling window of the last 180 blinks, 180 because it was making 2 classifications/predictions per second, by using a queue data structure. If the queue size exceeds 180, the oldest blink is removed from the queue, and the blink counters are updated accordingly. At the bottom its calculated the proportion of long blinks in the last 180 blinks and it is compared to a threshold of 25%, rule 1. If the proportion is higher, the *alert_var* variable is set to true. Also checks if there have been 4 or more consecutive long blinks, rule 2. If this is the case, the *alert_var* variable is set to true. Ending the listing the final line returns the value of the *alert_var* variable, indicating whether an alert should be generated.

```
bool alert_var = 0;

// Rules verification

    if (output_y == 2) {
        long_blinks++;
        consecutive_long_blinks++;
    }else if(output_y == 1){
        short_blinks++;
        consecutive_long_blinks = 0;
    }else{
        consecutive_long_blinks = 0;
    }
    total_blinks++;

    blink_queue.push_back(output_y);

    if (blink_queue.size() > 180) {
        int oldest_blink = blink_queue.front();
        blink_queue.pop_front();

        if (oldest_blink == 2) {
            long_blinks--;
        }else if(oldest_blink == 1){
            short_blinks--;
```

```
26          }
27          total_blinks--;
28      }
29
30      double proportion = (double)long_blinks / total_blinks;
31
32      if (proportion > 0.25 ) {
33          alert_var = true;
34      }else{
35          alert_var = false;
36      }
37      if (consecutive_long_blinks >= 4) {
38          alert_var = true;
39      }
40
41  return alert_var;
```

Listing 42: Drowsiness Detection Rules.

Last but not least, the second function is *getAlertNumber*, Listing 43. Returns the value of a member variable *alertNumber*. Which is the number of times the driver was alerted.

```
1  return this->alertNumber;
```

Listing 43: Drowsiness Detection Alerts.

6.12 Database

As previously stated, this system's database is developed using MySql. One can create a file, *D3_database.sql* to have the queries needed to create the database. Analysing the two main entities of the database, user and trip, one can see its implementation in Listing 44.

```
1  --
2  -- Database: `D3_database
3  --
4
5  -- --------------------------------------------------------
6  -- Create database
7
8  DROP DATABASE IF EXISTS `D3_database`;
9  CREATE DATABASE `D3_database`;
10 USE `D3_database`;
11
12 -- --------------------------------------------------------
13
14 --
15 -- Structure of the `User` table
```

```
--

DROP TABLE IF EXISTS `User`;
CREATE TABLE `User` (
  `USER_ID` int NOT NULL AUTO_INCREMENT,
  `name` char(50) NOT NULL,
  `email` char(50) NOT NULL,
  `password` char(50) NOT NULL,
  PRIMARY KEY (`USER_ID`)
);

-- --------------------------------------------------------

--
-- Structure of the `Trip` table
--

DROP TABLE IF EXISTS `Trip`;
CREATE TABLE `Trip` (
  `TRIP_ID` int NOT NULL AUTO_INCREMENT,
  `trip_duration` float(255,0) NOT NULL,
  `number_of_alerts` int NOT NULL,
  `USER_ID` int DEFAULT NULL,
  PRIMARY KEY (`TRIP_ID`),
  CONSTRAINT `USER_ID` FOREIGN KEY (`USER_ID`) REFERENCES `
    User` (`USER_ID`)
               ON UPDATE CASCADE
);
```

Listing 44: Queries to create tables user and trip.

In table User there are a set of attributes needed to identify a particular user, such as name, email and password. Each User instance is directly connected with the existence of a Trip. That way, the primary key for the Trip table references the primary key from User, which is the *USER_ID*. The foreign key constraint is set with the "ON UPDATE CASCADE" option, which means that if a row in the User table is updated, all corresponding rows in the Trip table that reference that *USER_ID* will be updated as well.

This creates a relationship between the User and Trip tables, where each row in the Trip table is related to one and only one row in the User table. In other words, each trip is taken by one user, and each user can take multiple trips.

One can execute all the queries in *D3_database.sql* in order to create the database by login into MySql and typing the command:

```
mysql> source ~/D3-driver-drownsiness-detection-master/
    D3_database.sql;
```

After this, its possible to see see the result shown in Figure **??**, in MySQL workbench.

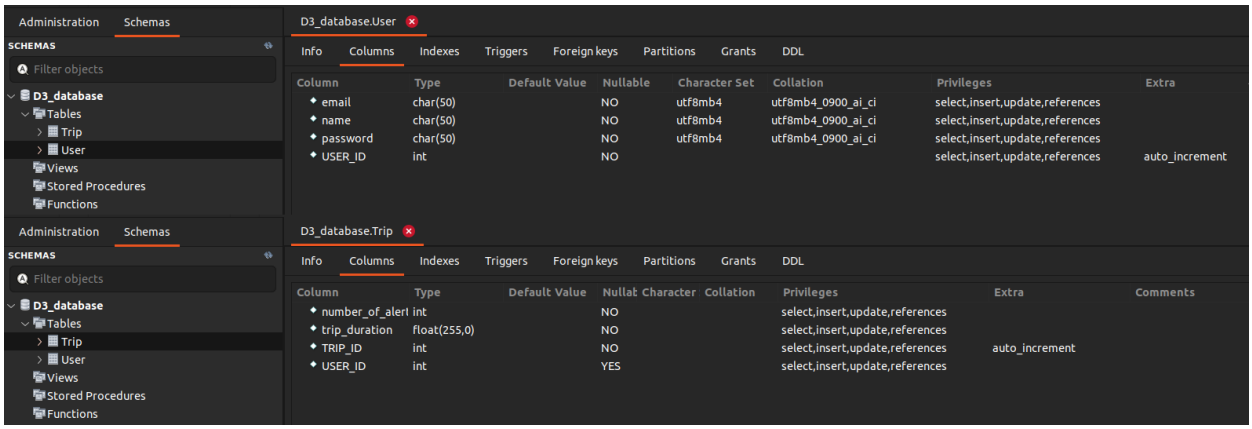


Figure 60: Database: created tables.

For the interaction of the database with the application it was written some PHP scripts that provide a set of functions to connect and execute SQL queries to MySQL database. First, to ensure that the code is always organized, a PHP script was created that provides some constants needed for interaction. In the listing 45, you can see the values assigned to each constant, corresponding to the database configuration.

```php
<?php
    define('DB_NAME','D3_database');
    define('DB_USER','root');
    define('DB_PASSWORD','');
    define('DB_HOST','localhost');
?>
```

Listing 45: Constants script

Before queries can be made, it is necessary to establish a connection to the database. To do this we used the function *mysqli* as shown in the listing 46.

```php
function connect(){
    include_once dirname(__FILE__).'/Constants.php';
    $this->con = new mysqli(DB_HOST,DB_USER, DB_PASSWORD,
   DB_NAME);
    if(mysqli_connect_errno()){
        echo "Failed".mysqli_connect_err();
    }
    return $this->con;
```

```
8 }
```

Listing 46: Database Connection.

To assist other scripts, a set of functions was implemented, defined in listing 47. Firstly, the *createUser* function checks whether the data for creating a new user already exists in the database, line 2. This ensures that each user has a unique username and email address. The *md5* is a widely-used hash function that takes an input and returns a 128-bit "hash" value. Afterwards it was used the *prepare* function to prepare a SQL statement for execution. This is a useful feature for preventing SQL injection attacks, as it allows you to bind values to placeholders in the SQL statement, rather than concatenating values into the string.

```
1  function createUser($username,$email,$pass)
2  {
3      if($this->isUserExist($username,$email)){
4          return 0;
5      }else{
6              $password = md5($pass);
7              $stmt = $this->con->prepare("INSERT INTO `User`(`
       USER_ID`, `name`, `email`, `password`) VALUES (NULL, ? , ?
        , ?);");
8              $stmt->bind_param("sss",$username,$email,
       $password);
9
10             if($stmt->execute()){
11             return 1;
12             }else{
13             return 2;
14             }
15         }
16 }
```

Similarly, the *insertTripData* function checks for the existence of the user, and runs an SQL statement to insert the trip data into the database.

```
17 function insertTripData($username,$email,$duration, $alerts,
       $userId){
18     if($this->isUserExist($username,$email)){
19         $stmt = $this->con->prepare("INSERT INTO `Trip`(`
       TRIP_ID`, `trip_duration`, `number_of_alerts`, `USER_ID`)
       VALUES (NULL, ? , ? , ?);");
20         $stmt->bind_param("sss", $duration, $alerts, $userId)
       ;
21
22         if($stmt->execute()){
23             return 1;
24         }else{
```

```
25              return 2;
26          }
27      }else{
28          return 0;
29      }
30  }
```

Regarding *userLogin*, this function performs the same encryption procedure used when creating a user and searches the User table, where the email and password match those entered. If the number of rows of the result is greater than zero, it means that the credentials are in the database, as shown in line 38.

```
31  function userLogin($email, $pass){
32      $password = md5($pass);
33      $stmt = $this->con->prepare(
34      "SELECT `USER_ID` FROM `User` WHERE `email` = ? AND `
    password` = ?");
35      $stmt->bind_param("ss",$email,$password);
36      $stmt->execute();
37      $stmt->store_result();
38      return $stmt->num_rows > 0;
39  }
```

Listing 47: Auxiliary functions

Finally, the scripts used to interact with HTTP requests will be demonstrated. Three scripts were implemented, one to register the user, another to perform the login, and another to transmit the travel data to the database. The three scripts are very similar and have the same procedure, so only the *registerUser* script will be explained. The next listing 48 shows the implementation of the script.

```
1  <?php
2
3  require_once ('../includes/DbOperations.php');
4
5  $response = array();
6
7  if($_SERVER['REQUEST_METHOD']=='POST'){
8      if(
9          isset($_POST['username']) and
10             isset($_POST['email']) and
11                 isset($_POST['password']))
12         {
13             $db = new DbOperations();
14
15             $result = $db->createUser(  $_POST['username'],
16                                         $_POST['email'],
```

```
17                                          $_POST['password']
18                                      );
19              if($result == 1){
20                  $response['error'] = false;
21                  $response['message'] = "User registered
    successfully";
22              }elseif($result == 2){
23                  $response['error'] = true;
24                  $response['message'] = "Some error occurred
    please try again";
25              }elseif($result == 0){
26                  $response['error'] = true;
27                  $response['message'] = "It seems you are
    already registered, please choose a different email and
    username";
28              }
29
30          }else{
31              $response['error'] = true;
32              $response['message'] = "Required fields are
    missing";
33          }
34 }else{
35      $response['error'] = true;
36      $response['message'] = "Invalid Request";
37 }
38
39 echo json_encode($response);
40 ?>
```

Listing 48: Register user script

Basically, an empty array is created and given the name response, where the response returned to the request will be stored. Then it is checked that the request method used to access the page is the post method, and it is verified that the various fields have been sent in the POST data, line 8 to 11. Finally, a user is created if everything is valid, and a response is returned to the page, which identifies whether the user creation was valid or not, line 39.

6.13 Mobile Application

As previously demonstrated, a mobile application has two stages of implementation. One focused on the graphical user interface, and another that focuses on the behavior that the program should have in the face of interaction. In this section, the process for creating the application with a professional interface and the desired behavior will be demonstrated in detail, that is, capable of establishing a Bluetooth connection with the local

system and communicating efficiently with the database server.

6.13.1 Bluetooth Communication

To establish Bluetooth communication between the mobile application and the local system, it is important that the necessary permissions are guaranteed. So it was used the method *checkSelfPermission* to guarantee such a thing, as the following listing demonstrates.

```
1 if (ContextCompat.checkSelfPermission(this,
    Manifest.permission.BLUETOOTH_CONNECT) != PackageManager.
    PERMISSION_GRANTED) {
2     ActivityCompat.requestPermissions(this, permissions,
    REQUEST_CODE_PERMISSIONS);
3     Log.i("Permission", "Permission request.");
4 }
```

To establish the connection to the local system and perform the writing and reading, the *BluetoothSingleton* class was implemented. This class was implemented using the Singleton design pattern, to ensure that only one object of the class is created. This class has some important methods for establishing bluetooth communication between devices, which are detailed below.

First, it is important to identify the devices paired to the mobile device, and identify if it is paired to the local system.

```
1  public boolean paired() {
2
3      boolean pairStatus = false;
4      pairedDevices = mBluetoothAdapter.getBondedDevices();
5
6      for (BluetoothDevice bt : pairedDevices)
7      {
8          if (Objects.equals(bt.getAddress(), myAddress))
9          {
10             pairStatus = true;
11         }
12     }
13     return pairStatus;
14 }
```

With the mac address of the local system already known (*myAddress*), all matched addresses are compared and if there is a match with the local system address, the method returns true.

Afterwards, it is necessary to accept connections made to the read socket channel, and to establish a connection to the local system for the write socket. Thus, the communication was split with two Bluetooth sockets. One that

handles data to be received and one that handles data to be sent from the application to the local system. These two methods will be called in a method of the *BluetoothSingleton* class that will run in a background thread, as the listing below demonstrates.

```
@Override
protected Void doInBackground(Void... params) {
    connectSocket();
    acceptSocket();
    while(true)
    {
        if(readFromLocal() == 1)
        {
            Log.e("Bluetooth","Error reading from local");
            break;
        }
        else{
            received = 1;
            Log.i("Bluetooth", "Msg Received");
        }
    }
    return null;
}
```

Also, the implementation of methods *acceptSocket* and *connectSocket* are demonstrated in the listings below.

The acceptSocket method, as the name implies, accepts connections through the channel chosen for the respective socket. On line 6, this method creates a Bluetooth listening socket with Service Logging, i.e. the system will assign an unused RFCOMM channel for listening and also log a Service Discovery Protocol (SDP) log with the local SDP server containing the specified UUID, service name, and self-assigned channel.

```
public short acceptSocket() {
    short error = 0;
    BluetoothServerSocket btSocketServer = null;

    try {
        btSocketServer = mBluetoothAdapter.
   listenUsingInsecureRfcommWithServiceRecord(device.getName
   (), MY_UUID);
    }catch (IOException e)
    {
        error = 1;
        Log.e("Bluetooth","Server socket failed");
    }

    try {
```

```
14          btSocketRcv = btSocketServer.accept();
15      }catch (IOException e){
16          error = 1;
17          Log.e("Bluetooth","failed to accept");
18      }
19
20      Log.i("Bluetooth","Connected");
21
22      try {
23          inpStream = btSocketRcv.getInputStream();
24      } catch (IOException e) {
25          errorExit("Error", "Get input Stream error: " + e.
    getMessage() + ".");
26          error = 1;
27          try {
28              btSocketRcv.close();
29          } catch (IOException e2) {
30              errorExit("Error", "Unable to close connection
    after connection failure: " + e2.getMessage() + ".");
31          }
32      }
33      return error;
34  }
```

In a similar way to the previous method, first a socket is created to establish the connection between the other device. For this the UUID of the serial port service is used, as shown in line 5. A connection is then initiated, line 11, and at the end, if the connection is made, an output stream is obtained for sending characters, line 21.

```
1  public short connectSocket()
2  {
3      short error = 0;
4      try {
5          btSocket = device.
    createInsecureRfcommSocketToServiceRecord(MY_UUID);
6      } catch (IOException e) {
7          errorExit("Error", "In onResume() happened the
    following error: " + e.getMessage() + ".");
8          error = 1;
9      }
10     try {
11         btSocket.connect();
12     } catch (IOException e) {
13         try {
14             btSocket.close();
15         } catch (IOException e2) {
16             errorExit("Error", "Unable to close connection
    after connection failure: " + e2.getMessage() + ".");
```

```
17              error = 1;
18          }
19      }
20      try {
21          outStream = btSocket.getOutputStream();
22      } catch (IOException e) {
23          errorExit("Error", "Stream creation failed: " + e.
    getMessage() + ".");
24          error = 1;
25      }
26
27      return error;
28  }
```

It then calls the read method,*readFromLocal* ,to handle the information received by the input stream from the receive socket, as is shown above.

```
1  public short readFromLocal() {
2      short error = 0;
3      try {
4          bytes = inpStream.read(buffer);
5          String readMessage = new String(buffer, 0, bytes);
6          Log.i("Read from local","Read from device: " +
    readMessage);
7          inputString = readMessage;
8          rcvFlag = 1;
9      } catch (IOException e) {
10         error = 1;
11         Log.e("Error","Read from device");
12     }
13     return error;
14 }
```

6.13.2 Database Communication

To interact with the database and send information from the application to it, a library called Volley was used. This library makes networking much more faster and easier for android applications. As already mentioned, the application will send information to the database, so the methods used for communication will now be defined.

First, as specified in the Analysis chapter, the user will have to log in, or create an account if he does not have one, to later transfer the data preserved in a file, to database. To register, a method called *registerUser* as shown in the listing below.

```
1  private void registerUser() {
2      final String username = editUser.getText().toString().
    trim();
```

```
3     final String email = editEmail.getText().toString().trim
      ();
4     final String password = editPass.getText().toString().
      trim();
5
6     if (TextUtils.isEmpty(username)) {
7         editUser.setError("Please enter username");
8         editUser.requestFocus();
9         return;
10    }
11
12    if (TextUtils.isEmpty(email)) {
13        editEmail.setError("Please enter your email");
14        editEmail.requestFocus();
15        return;
16    }
17
18    if (!android.util.Patterns.EMAIL_ADDRESS.matcher(email).
      matches()) {
19        editEmail.setError("Enter a valid email");
20        editEmail.requestFocus();
21        return;
22    }
23
24    if (TextUtils.isEmpty(password)) {
25        editPass.setError("Enter a password");
26        editPass.requestFocus();
27        return;
28    }
```

First, a request is started, in which the data obtained in the user's fields is sent as a parameter. From line 6 to 28 checks are performed to ensure that the data is valid. Then an object of type *StringRequest* is created, which is used to add a network request to the queue for processing - line 63.

```
29    StringRequest stringRequest = new StringRequest(Request.
      Method.POST, URLs.URL_REGISTER,
30        new Response.Listener<String>() {
31            @Override
32            public void onResponse(String response) {
33                try {
34                    JSONObject obj = new JSONObject(response)
      ;
35                    if (!obj.getBoolean("error")) {
36                        Toast.makeText(getApplicationContext
      (), obj.getString("message"), Toast.LENGTH_SHORT).show();
37                        finish();
38                        startActivity(new Intent(
      getApplicationContext(), LoginActivity.class));
```

```
39                      } else {
40                          Toast.makeText(getApplicationContext
    (), obj.getString("message"), Toast.LENGTH_SHORT).show();
41                      }
42                  } catch (JSONException e) {
43                      e.printStackTrace();
44                  }
45              }
46          },
47          new Response.ErrorListener() {
48              @Override
49              public void onErrorResponse(VolleyError error) {
50                  Toast.makeText(getApplicationContext(), error
    .getMessage(), Toast.LENGTH_SHORT).show();
51              }
52          }) {
53          @Override
54          protected Map<String, String> getParams() throws
    AuthFailureError {
55              Map<String, String> params = new HashMap<>();
56              params.put("username", username);
57              params.put("email", email);
58              params.put("password", password);
59              return params;
60          }
61      };
62
63      VolleySingleton.getInstance(this).addToRequestQueue(
    stringRequest);
64  }
```

This request will be processed by a background thread provided by the Volley library. It is also important to note, that associated with the network request is specified the *onResponse* method (line 32), which retrieves the response body at a given URL as a string. In this way, it is verified if the response received is valid, and if so, the *LoginActivity* activity is initiated, as shown in line 38.

Regarding the application login, a method quite similar to the previous one was used. A network request is created again, but the URL used is the one corresponding to the location of the login php script.

After login, it is now possible to send the trip data to the database. As is known, the user must provide the application with a file in csv format, downloaded by the application at the end of each trip. The system will have to read the information in this file and send the *trip_duration* and *number_of_alerts* fields to the database. The method used is very similar to the previous ones, with only the parameters sent with the request varying.

6.14 Mechanical Structure

In this section the implementation of the mechanical structure will be analyzed. The mechanical structure was completely made using 3D printing.

The 3D design was started in the Fusion 360 program and the final results are as shown in the following Figures. This structure is composed by a base, a cover and a camera and support parts.

6.14.1 The Base

This support, seen in Figure 61, is the base for D3. This support has slightly larger dimensions than the raspberry pi and fits perfectly.

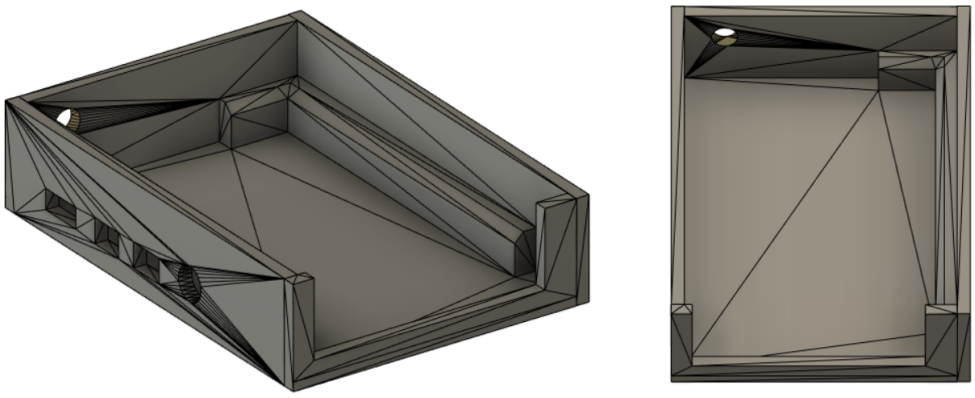


Figure 61: D3 base.

6.14.2 The Cover

This support, seen in Figure 62, is the cover for D3. It has the same dimensions as the base in terms of length and height. It contains an opening for the camera connection and contains the first support that holds the camera's tilt.

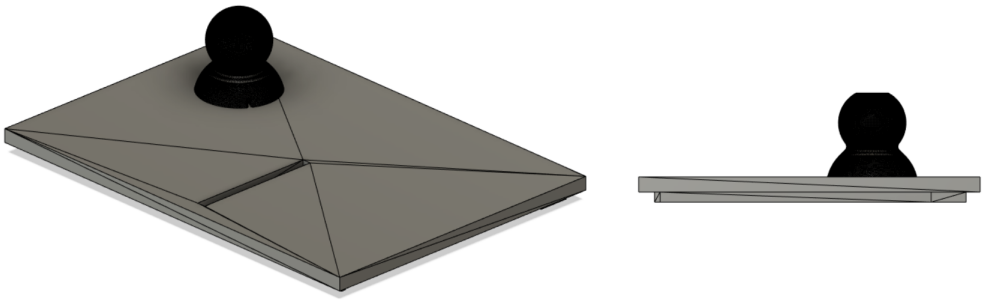


Figure 62: D3 cover.

6.14.3 Camera and Support

The camera support seen in Figure 63 holds the camera. It is also possible to observe the parts that work as cover and protection of the camera.

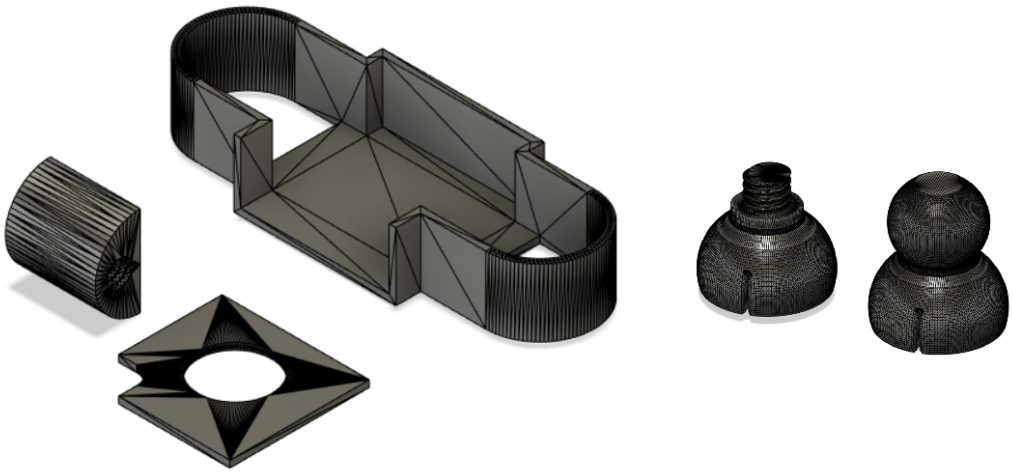


Figure 63: D3 Camera and Support parts.

6.15 Final Product: D3

Now that all the parts are completed it is possible to assemble everything in order to obtain the final product of D3.

In the Figure 65 it is possible to see the assemble of D3.

The Figures you can see below shows the D3 from several angles, for a better illustration of what D3 really is.

Figure 64: D3: as final product

Figure 65: D3: as final product

6.15.1 How to it works

Before powering up the system, it is necessary to ensure that it is correctly positioned and fixed in place to place the product. In this way, the system must remain immobile to any noise resulting from the movement of the car, and the camera inclination must be such that it allows the system to detect the user's eyes with perfect clarity, as shown in Figure 66.

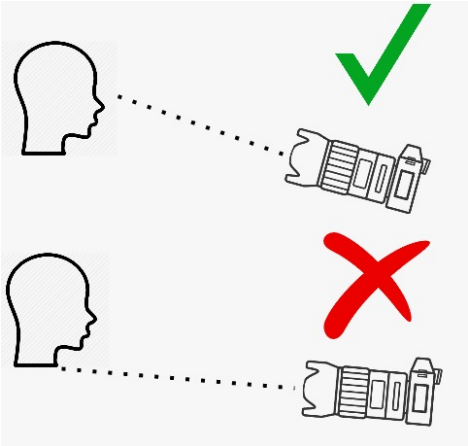


Figure 66: Correct camera position

Finally, the user must connect a sound device to the system Aux input, and power it up. At the end of this process, the device is ready to be used.

7 Verification Tests

7.1 Camera

To determine effectively whether or not the camera was detected by the Raspberry Pi, it was used the command *vcgencmd get_camera*, and the result was supported=1 detected=1, showing that the camera was successfully detected. Next, in order to test the camera's behavior, a small code was written to visually check the image quality, Listing 49.

```
#include "/usr/include/opencv2/opencv.hpp"
#include "/usr/include/opencv2/face.hpp"

using namespace std;
using namespace cv;
using namespace cv::face;

#include <unistd.h>
#include <iostream>

#define IMAGE_PATH  ("/root/testframe.jpg")

#define FRAME_W     640
#define FRAME_H     480

#define CAM_BRIGHTNESS  0.5
#define CAM_CONTRAST  0.5
#define CAM_SATURATION  0.5

#define deviceID 0

int main(int argc,char** argv)
{

cv::VideoCapture cam;
cam.open(deviceID, cv::CAP_V4L2);


/******** Camera Configurations **********/
cam.set(CAP_PROP_FRAME_WIDTH , FRAME_W);
cam.set(CAP_PROP_FRAME_HEIGHT, FRAME_H);

cam.set(CAP_PROP_BRIGHTNESS, CAM_BRIGHTNESS);
cam.set(CAP_PROP_CONTRAST, CAM_CONTRAST);
cam.set(CAP_PROP_SATURATION, CAM_SATURATION);

system("v4l2-ctl --set-ctrl=auto_exposure=0");
system("v4l2-ctl --set-ctrl=iso_sensitivity=4");
```

```
39 system("v4l2-ctl --set-ctrl=auto_exposure_bias=18");
40 system("v4l2-ctl --set-ctrl=exposure_time_absolute=1000");
41 system("v4l2-ctl --set-ctrl=exposure_dynamic_framerate=1");
42 system("v4l2-ctl --set-ctrl=white_balance_auto_preset=7");
43 system("v4l2-ctl --set-ctrl=sharpness=30");
44
45 Mat frame;
46
47 cam.read(frame);
48
49 if (frame.empty())
50     printf("[Camera] blank frame captured.\n");
51
52 imwrite(IMAGE_PATH, frame);
53
54 cam.release();
55
56 return 0;
57 }
```

Listing 49: Camera Test.

The procedure used to verify that the camera was operational and mostly of good quality was quite simple. The program shown in listing 49 was run, which captures a frame from the camera and stores the frame in the same directory as the program. Then the command *scp -r /root/testframe.jpg andre@10.42.0.1: /Desktop* was executed, which copied that same frame to the work computer desktop. Thus, the quality of the camera was visually checked, and the parameters were adjusted until the resulting frame was equal to the one specified in Figure 67.

Figure 67: Camera Tests (Frame before calibration, after, and the night frame respectively).

7.2 Eye Landmarks Detection

Regarding the eye landmarks detection, it was done as previously mentioned in the implementation. First, it was tested the algorithm that detects the face, and next the algorithm that detects the 68 landmarks of the face, but in the tests below only was highlighted the eye landmarks which are the ones that really matter. In Figure 68 is shown the result of the eye landmarks outline detection.

As one can see, the algorithm detected the all the coordinates that compose the eyes.

Figure 68: Eye Landmarks Detection Test (open eyes, and closed eyes respectively).

7.3 SVM Accuracy

As already explained, the drowsiness detection algorithm is the most important algorithm in the system, and its effectiveness is crucial to the success of the project. Thus, to test its effectiveness, the system will be placed to evaluate a set of people, whose state of sleepiness was rigorously evaluated by the University of Liège [19], and provided the videos to allow an evaluation of sleepiness detection systems. In order to identify the person's state of sleepiness, this study followed a table represented in Figure 69.

According to Figure 69, lower KSS levels correspond to situations where the subjects do not have reduced performance due to drowsiness. Higher KSS levels represent possible states of sleepiness. Various studies analyzed [20]

Rating	Verbal descriptions
1	Extremely alert
2	Very alert
3	Alert
4	Fairly alert
5	Neither alert nor sleepy
6	Some signs of sleepiness
7	Sleepy, but no effort to keep alert
8	Sleepy, some effort to keep alert
9	Very sleepy, great effort to keep alert, fighting sleep

Figure 69: KSS scale for auto-evaluation of drowsiness.

several thresholds for KSS class categorization, and the best performance for mapping KSS in their SVM model was related to KSS ≤ 3 for alertness and KSS ≥ 7 for drowsiness. The middle state is designated by neutral state.

Following the classification obtained for each video, the camera was placed to capture frames of the provided video and the number of alerts detected for each was recorded, Figure 70.

Figure 70: Drozy Database Videos Tests.

The results are represented in table 10.

Subject	**Day**	**KSS**	**Warnings emitted**
1	3	7	2
2	3	9	4
3	3	9	42
4	1	2	0

Table 10: KSS classification and number of warnings emitted considering four subjects.

7.4 LED Device Driver

To test the implementation of the LED device driver it was wroted the code described in Listing 50. Was created with the purpose of testing the behavior of the green LED and this program makes use of a device driver for a GPIO pin.. The purpose of this code is to make use of the device driver and perform a kind of "light", that is, it activates the sound for 1 second, turns off 1 second and the cycle is repeated.

```
58 #include <unistd.h>
59 #include <fcntl.h>
60 #include <iostream>
61
62 int main (void){
63
64 char LedOn = '1';
65 char LedOff = '0';
66
67 system("insmod led.ko");
68 int file_descriptor = open("/dev/led0", O_WRONLY);
69
70 write( file_descriptor , &LedOn, 1);
71 sleep (1);
72
73 write( file_descriptor , &LedOff, 1);
74 sleep (1);
75
76 close ( file_descriptor );
77 system("rmmod led.ko");
78
79 return 0;
80 }
```

Listing 50: LED Driver Test.

One can see in Figure 71, that the LED lights up after running the program, thus verifying the correct implementation of the device driver.

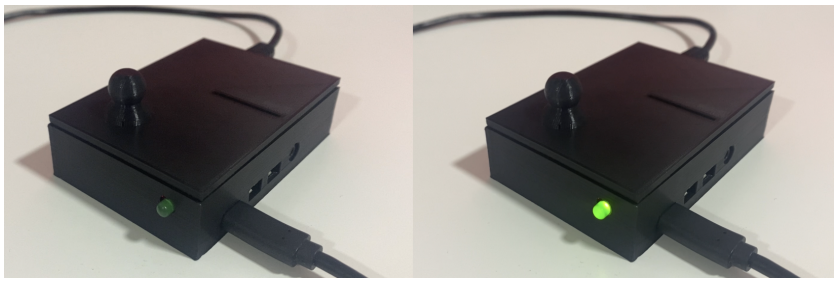


Figure 71: Test green LED: led-test

7.5 Audio

To check the correct operation the command *speaker-test* was executed The *speaker-test* command allowed us to verify that the audio was being played, Figure 72.

```
# speaker-test

speaker-test 1.2.4

Playback device is default
Stream parameters are 48000Hz, S16_LE, 1 channels
Using 16 octaves of pink noise
Rate set to 48000Hz (requested 48000Hz)
Buffer size range from 512 to 65536
Period size range from 512 to 65536
Using max buffer size 65536
Periods = 4
was set period_size = 16384
was set buffer_size = 65536
 0 - Front Left
Time per period = 1.383481
 0 - Front Left
Time per period = 2.729980
 0 - Front Left
```

Figure 72: Test audio: speaker-test

7.6 Database

Regarding the database, to check if a PHP script is operational and the values are being correctly stored, the Postman tool was used.

First the registration of a user in the database was checked. To do this we specified the URL of the registration script and filled in the fields referring to the user to be created, as shown in Figure 73;

As you can see in the figure, the message returned indicates that the user has been successfully registered. Likewise, in Figure 74 you can see that the user was created in the database with the correct field values.

Afterwards, the Postman tool was used again to verify a user's login. As can be seen in Figure 75 the message received on the page indicates that the user is registered, and can be logged in.

Also, it has been checked that the trip data is being correctly sent to the database. In the same way as before, the Post method was sent with the respective fields filled in and the message in Figure 76 was received, indicating that the data was successfully saved in the database.

In Figure 77 you can see that the data sent via Postman corresponds to the data stored in the database. Thus, it is verified that all the behavior of data transmission to the database is correctly implemented.

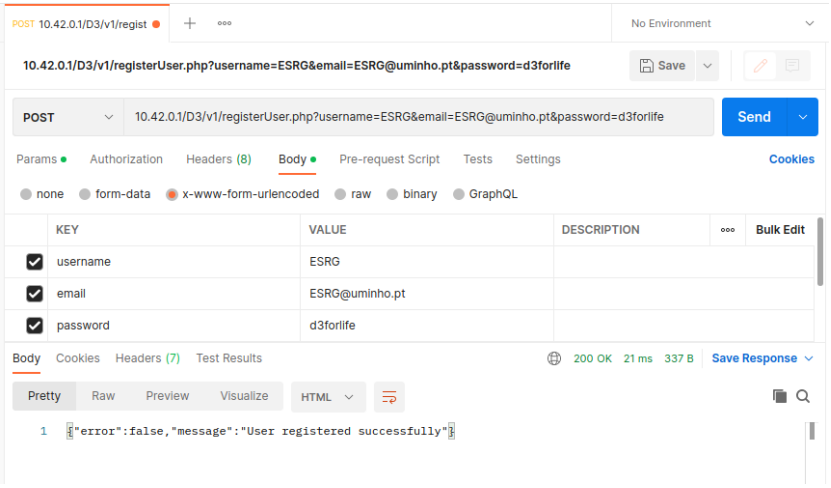


Figure 73: Test database: user registration

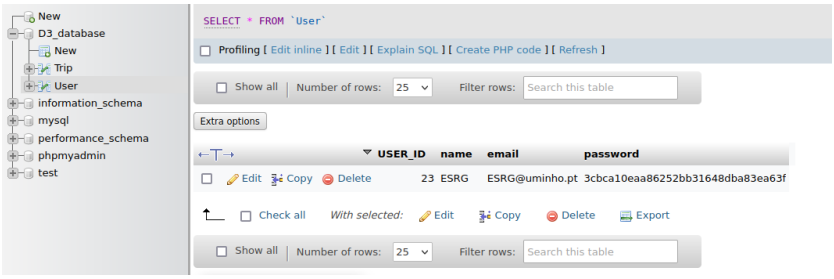


Figure 74: Test database: user registration

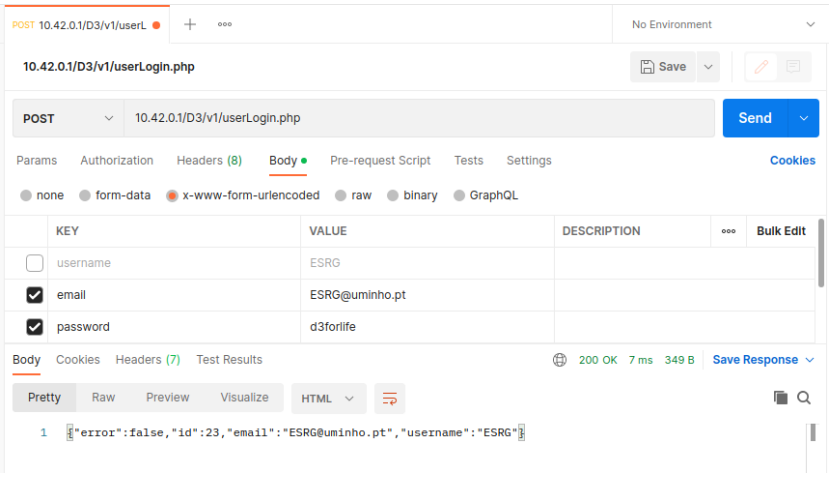


Figure 75: Test database: user login

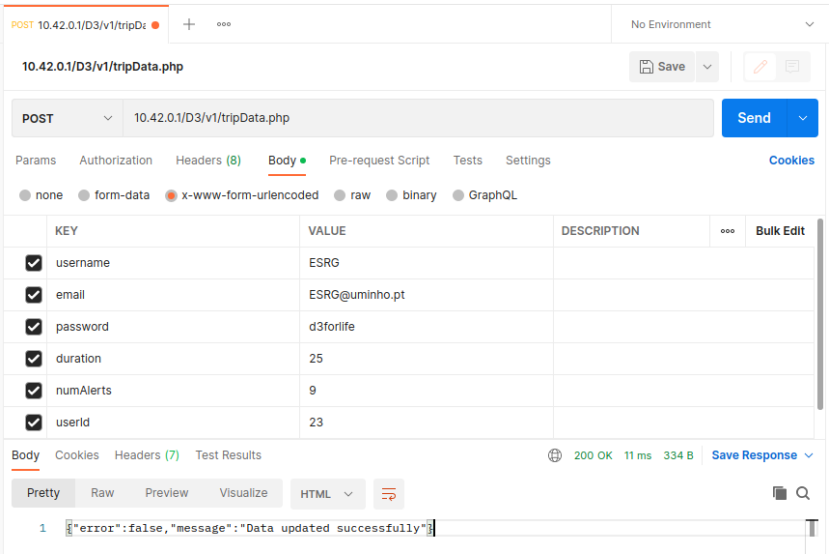


Figure 76: Test database: data trip storage

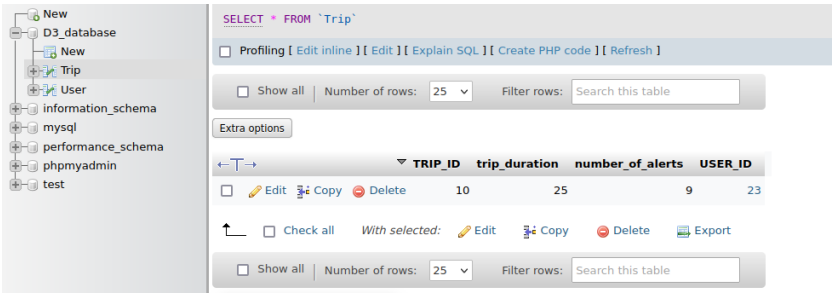


Figure 77: Test database: data trip storage

7.7 Mobile Application

7.7.1 Bluetooth Communication

Regarding the communication via Bluetooth, an objective code was written in order to verify that the connection, sending and receiving of characters was being carried out correctly between the application and the local system.

So, the sending of characters to the mobile application was tested, with the code specified by listing 51.

```
81 char dataSend[] = "1,7,";
82 status = ptr->m_sendBlue.connectToRemote();
83 if (status == 0)
84 {
85     status = ptr->m_sendBlue.sendToRemote(dataSend, 5);
```

```
86 }
```

Listing 51: Mobile application test: bluetooth

In this code, functions from the *Bluez* library were used to connect with the remote device and send a specific message. The message sent from the system to the mobile application was , represented in line 1. In the console of the android studio tool, it was possible to observe the successful connection between devices and the correct reception of the message, as shown in Figure 78.

```
I/Bluetooth: Connected
I/Read from local: Read from device: 1,7,
I/Bluetooth: Byte Received
```

Figure 78: Test mobile application: Bluetooth communication

7.7.2 Database

In this section three tests will be performed on the developed application:

- Creating an account;
- Login;
- Send the data of a trip, stored in a file, to the database.

This way, the developed application was initialized and the window for registering a user was opened, as shown in Figure 79.

In the fields to be filled in, the data for registration was entered and the registration button was pressed. As can be seen in Figure 80 and 81, the registration was successful, and the information was stored in the database.

In the same way, and following the previous procedure, the fields in Figure 81 were filled in, this time to perform the login. After pressing the Login button, it was verified in the Figure 82 that the login was successful, and the user is now ready to send data to the application.

Thus, the user is required to select a file, and after selection a success message is returned to the user, and consequently the data is stored in the database, Figure 83.

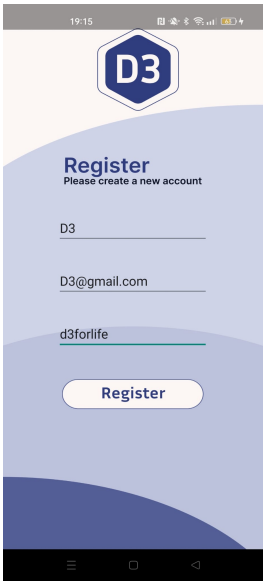


Figure 79: Test mobile application: register in database

Figure 80: Test mobile application: successful registration

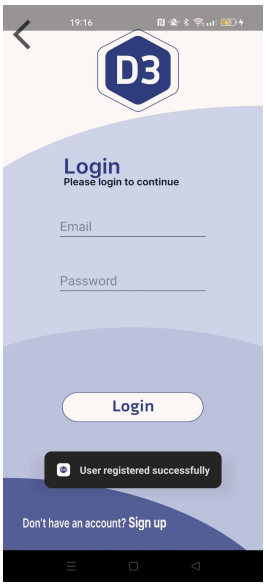


Figure 81: Test mobile application: Login in database

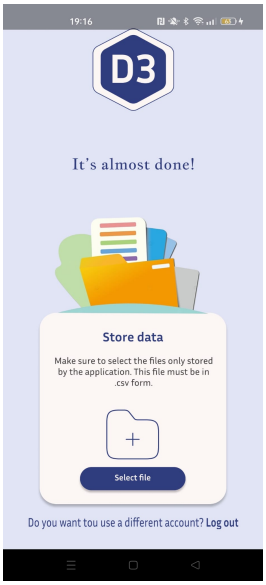


Figure 82: Test mobile application: Transfer to database

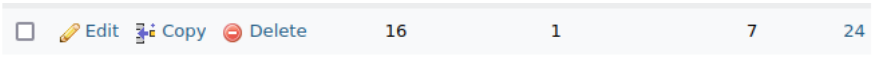


Figure 83: Test mobile application: successful transfer

7.8 Unit Test Cases

7.8.1 Daemon

The Daemons test cases performed are exposed on table 11.

Unite Test cases	Expected Results	Real Results
Create Daemon	Create Daemon and starts executes its own code	As expected
Signals	Receive from main program and handle them	As expected
Functions execution	Execute the property functionaccording to the signal received	As expected
Message Queue Creation	Create a message queuenon-blocking with specificsize and number of elements parameters	As expected
Delete Message Queue	Delete a message queue from the system	As expected
Message Queue communication	Send and receive data betweenthe daemon and the main program in a non-blocking away	As expected

Table 11: Daemons Unit Test Cases.

7.8.2 Threads

The threads test cases performed are exposed on table 12.

Unite Test cases	Expected Results	Real Results
T_BluetListening	Receive commands from the remote system, via Bluetooth, to initiate the trip	As expected
T_BluetTransmission	Transmits to the remote system, via Bluetooth, the the number of times the alert was activated and the trip duration	As expected
T_CamCapture	Perform constant frame captures. Execute to obtain points around the eye. Calculate the EAR of this frame. Flag the thread T_CamProcess	As expected
T_CamProcess	Analyze the N frame captures that are in shared memory, whenever it is flagged. Return set of frames there was a short blink, long blink or the eye is open. Conclude whether it needs to alert the driver or not	As expected
T_Alert	Alert signal received by the daemon. Executes and performs the necessary sound processing to alert the driver	As expected

Table 12: Threads Unit Test Cases.

7.8.3 Device Driver

The device driver test cases performed are exposed on table 13.

Unite Test cases	Expected Results	Real Results
GPIO register configuration and action	Configure PIN as output and change its value to HIGH and LOW	As expected
Read function	Receive the command	As expected
Read function	Actuate the right pin accordingly.	As expected

Table 13: Device Driver Unit Test Cases.

7.8.4 Database

The database test cases performed are exposed on table 14.

Unite Test cases	Expected Results	Real Results
Open database	Database open successfully	As expected
Send data to database	Database saves data correctly and successfully	As expected
Receive data from database	The database sends the requested data	As expected
Foreign keys	Delete and Update other line table that depends each other	As expected

Table 14: Database Unit Test Cases.

7.8.5 Mobile Application

The mobile application test cases performed are exposed on table 15.

Unite Test cases	Expected Results	Real Results
Register account	Update database with new account	As expected
Login into account	Get the login credentials from the database and make a comparison with them	As expected
Logout of account	Logout valid	As expected

Table 15: Mobile Application Unit Test Cases.

7.9 Integration Cases

The integration test cases performed on the system are presented on table 16.

Unite Test cases	Expected Results	Real Results
Open Eye	Camera detects and predicts the open eye	As expected
Short Blink	Camera detects and predicts the short blink	As expected
Long Blink	Camera detects and predicts the long blink	As expected
Drowsiness detection Rule 1	Emits an sound message and alerts the driver	As expected
Drowsiness detection Rule 2	Emits an sound message and alerts the driver	As expected

Table 16: Integration test cases.

8 Budget

The table 17 includes the final total price. Comparing the final with the estimated budget, some components were changed and others were added. It should be noted that all this modifications was already updated in the design phase. The price considered for the manpower hours was 10 euros/h. Considering all these points, we will have a total development cost of D3 around 4120 euros.

Name	**Estimated Cost (€)**
Raspberry Pi 4	60
Camera Module	40
Actuators Module	10
Structure	10
André 200 hours	2000
Carlos 200 hours	2000
Total	4120

Table 17: Final Budget.

9 Conclusion & Future Work

The goal of this project was to be able to develop a device that would be reliable in detecting drowsiness and allow its users to store the information after the trip in a database for the owner of the product, to be able to analyze the driving of different users.

Firstly, machine learning was used to identify patterns that could more reliably detect high levels of drowsiness in the driver. The model used was exposed to several tests for different degrees of drowsiness, and was able to distinguish between situations of extreme drowsiness and situations in which it was not necessary to alert the user. However, it was found that as the user moves away from the camera, the system is unable to accurately detect the driver's EAR values, and thus, the model's effectiveness is lower. It was also found that the position of the camera relative to the user, more specifically the degree of tilt with the positioning of the driver's face, also affects the effectiveness of the model. Most of the time this problem does not occur, but also with several tests it was detected that the luminosity also plays a role in the correct functioning of the system. During the day as long as there is good lighting the system works well, and at night thanks to the night vision there are almost no false positives either. But in environments where there is light but in a reduced form the detection ends up not being as accurate as it should be.

Thus, it is clear that there is still a great deal of room for improvement to ensure that the system is able to provide effective results for any usage scenario. One of the starting points could be the use of a different camera, one with a zoom setting and higher resolution quality when capturing frames.

Another point would be the detection time. Tests have shown that the alerts come with a delay of about two seconds after the driver's eyes are closed. These delays are caused by several elements of the system. One of the main ones would be the time needed by the model to predict, and as analyzed, the use of the OpenCV library to implement the model, despite being easier to implement, the speed and memory efficiency are lower than the Dlib library. In this way, the use of the Dlib library can reduce these delay times, and also ensure greater efficiency in detection.

More noteworthy future works, a good inclusion was to include monitoring of other facial landmarks (e.g. mouth landmarks) to improve the detection method and avoid false positive drowsiness state when the user yawns, for example. To conclude the application is physically non-intrusive and could be run alongside other programs in a consumer class computer without noticeable practical performance impact.

In addition, here is a link to a demonstration, (`https://youtu.be/`

1eHUM57p-ko) and to the source code (https://github.com/andrebarbosaa/D3-driver-drownsiness-detection.git).

10 Bibliography

[1] Texas Department of Insurance, “Driving Distractions Fact Sheet”, Apr 2020.

[2] I. Schagen, “Road Safety Thematic Report - Fatigue”, Jan 2021.

[3] C. Goldenbeld, D. Nikolaou, “Driver Fatigue”, Fev 2022.

[4] D. Costa, “A fadiga na condução”, Nov 2014.

[5] “Global driver drowsiness detection system market growth insights 2022”, (https://www.marketwatch.com/press-release/global-driver-drowsiness-detection-system-market-growth-insights-2022-size-research-latest-opportunities-top-growing-factors-key-dynamic-analysis-development-emerging-technologies-and-share-forecast-to-2029-2022-08-28)

[6] “Automotive driver drowsiness detection system market”,(https://www.fortunebusinessinsights.com/automotive-driver-drowsiness-detection-system-market-103620)

[7] “CIPIA-FS10”,(https://fs10.cipia.com/)

[8] SEIDL, Martina; SCHOLZ, Marion; HUEMER, Christian; KAPPEL, Gerti. UML @ Classroom, An Introduction to Object-Oriented Modeling. Springer International Publishing AG, 2015.

[9] KERRISK, Michael. The Linux Programming Interface, No Starch Press, 2010.

[10] VENKATESWARAN, Sreekrishnan. Essential Linux Device Drivers, Prentice Hall, March 27, 2008.

[11] CPU affinity ,(https://www.redhat.com/sysadmin/tune-linux-tips)

[12] Maior C.B.S., das Chagas Moura M.J., Santana J.M.M., Lins I.D. Real-time classification for autonomous drowsiness detection using eye aspect ratio

[13] Stern, J. A., Boyer, D. J., Schroeder, D. J., Touchstone, R. M., Stoliarov, N. (1996). ”Blinks, saccades, and fixation pauses during vigilance

task performance: II. Gender and time-of-day". Washington DC

[14] Gornik D., "Entity Relationship Modeling with UML", Nov 2013.

[15] (https://docs.opencv.org/3.4/d1/d73/tutorial_introduction_to_svm.html)

[16] (http://www.jiansun.org/papers/CVPR14_FaceAlignment.pdf)

[17] (https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/)

[18](https://www.sciencedirect.com/science/article/abs/pii/S0957417420303298)

[19] Massoz, Q., Langohr, T., Francois, C., Verly, J. G. (2016). The UL multimodality drowsiness database (called DROZY) and examples of use. In 2016 IEEE winter conference on applications of computer vision, WACV 2016 (https://doi.org/10.1109/WACV.2016.7477715)

[20]Ogino, M., Mitsukura, Y. (2018). Portable drowsiness detection through use of a prefrontal single-channel electroencephalogram. Sensors (Switzerland).