Federal University of Technology - Paraná – UTFPR
Academic Department of Electronics – DAELN
Academic Department of Informatics – DAINF
Computer Engineering
Integration Workshop 3 (EEX23) – S71 – 2023/1

Technical Report
AutoFleet - A solution to prevent accidents caused by drowsiness drivers

Eduardo Piaceny Ribas – eduardoribas@alunos.utfpr.edu.br
Hadryan Salles – hadryansalles@alunos.utfpr.edu.br
Ianca Polizelo – iancapolizelo@alunos.utfpr.edu.br
João Victor Laskoski – laskoski@alunos.utfpr.edu.br

June, 2023

Abstract

Brazil is a country that heavily uses the road network as its main mean of transport. Therefore, there are many drivers and also fleets that use this means. Often, transport workers have long hours of work, without taking adequate breaks for rest, which ends up causing several traffic accidents due to the fact that they lose attention and even fall asleep at the wheel. With that in mind, this project seeks to implement an intelligent solution that can prevent traffic accidents caused by drowsiness. Through a device installed in the vehicle, it is possible to monitor the driver's face and hands to ensure that he has a smooth and safe trip. GPS and accelerometer also keep track of vehicle position, speed and acceleration. If the driver happens to get sleepy, the device will emit sound alerts with the intention of waking him up. Furthermore, all collected data is periodically sent to a cloud server so that the fleet manager can take care of his employees and make better decisions.

1 Introduction

Traffic accidents caused by drowsiness present a concerning problem within the brazilian traffic context, with substantial implications for road safety and public well-being. According to available data, car accidents related to sleeping while driving have been alarmingly prevalent in Brazil. In fact, a study conducted by the Brazilian Association of Road Traffic Medicine (Associação Brasileira de Medicina de Tráfego - ABRAMET) [1] revealed that drowsiness contributed to approximately 42% of all recorded traffic accidents in Brazil.

Thus, the idea of this project is to help in minimizing this problem, and improve the working conditions of drivers by building a device that is able to monitor them and detect when they are presenting symptons of drowsiness, making it possible to issue alerts to wake them up.

In addition, the device has other features such as: detecting whether the driver's hands are touching the steering wheel, measuring vehicle speed, position and acceleration, while storing all the periodically captured data during the trip.

1.1 Overview

Figure 1 shows the Block Diagram of the system. As can be seen, the device comprehends an embedded system which monitors all the sensors and actuate on the sound. On top of that, using network connection, the device communicate with a webserver sending data packets with the collected data from sensors. Besides storing the data, the webserver also provides a website to visualize trip details.

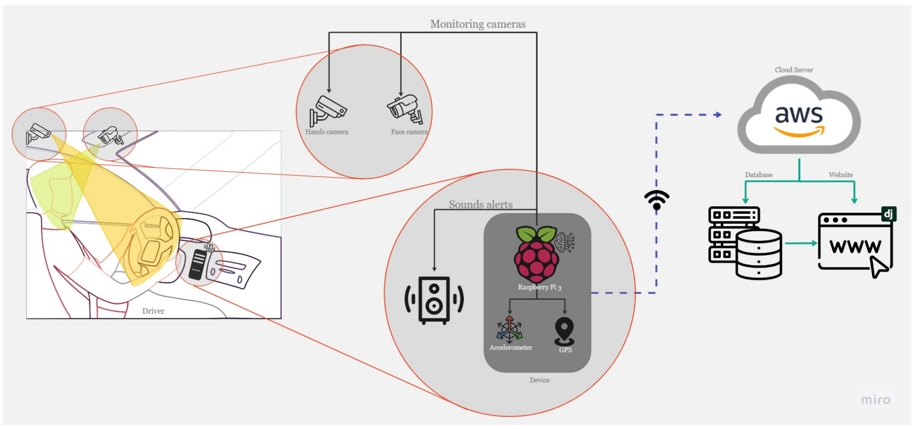


Figure 1: Block Diagram

The embedded system comprises a Raspberry Pi3 B linked to multiple components. These components include two USB cameras affixed to the car's sun visor, providing visibility of the driver's face and the steering wheel. Additionally, there is a GPS for tracking the vehicle's location during the entire trip, a 3-axis accelerometer to measure acceleration changes, and a speaker to emit sound alerts, ensuring the driver is awakened if they happen to present drowsiness while driving. Lastly, the device is equipped with an SD card to store data locally in cases of connection loss, aiming to send this data once the connection is established.

The website serves two user groups: Drivers and Managers, each offering distinct features. For Drivers, the platform enables them to initiate a trip by logging in and selecting a designated button, notifying the server to associate all subsequent data with that specific trip. On the other hand, Fleet Managers have the capability to retrieve and analyze comprehensive trip histories from all drivers within the fleet.

2 Project Specification

The project was separated into three main components, Mechanical, Hardware and Software, with their requirements presented in the sections 2.1, 2.2 and 2.3 respectively. It was also necessary to create anti-requirements that are presented in the section 2.4.

2.1 Mechanical Functional Requirements

- **MFR1** - The device's mechanical structure must attach together the microprocessor, cameras, GPS, accelerometer and buzzer in a box of dimensions 18x7x3 cm, made of MDF (Medium-Density Fibreboard) wood where the electronic boards are fixed.
- **MFR2** - The device's mechanical structure must provide safety from the sun for all electronic components using a thermal insulating material such as wood.
- **MFR3** - The device's mechanical structure must be solid to protect the components against accelerations caused by holes in the highway or other shakes in the vehicle.
- **MFR4** - The face camera must be positioned so that it has a clear vision of the driver's face.
- **MFR5** - The hand camera must be positioned facing below so it has a clear vision of the steering wheel.
- **MFR6** - Both camera cables must be connected to the device without blocking the driver's vision of the road.
- **MFR7** - Both cameras must be fixed in the car sun-protector using clip.
- **MFR8** - Both camera cables must be fixed in the side of the car using double-sided tape.
- **MFR9** - The device mechanical structure must be positioned on the car's panel without blocking the driver's vision of the road, using an 11-inch smartphone support for this.

2.2 Hardware Functional Requirements

- **HFR1** - The hardware must detect acceleration changes in the vehicle with precision of at least 0,1 m/s^2 in three axes, using an MPU-6050 accelerometer sensor with ±4g range.
- **HFR2** - The hardware must be turned off whenever the vehicle is turned off to make possible to detect the moments when the hardware is turned off.

- **HFR3** - The hardware must emit sound alerts using a buzzer when the output of the image processing of the driver is “drowsy”.
- **HFR4** - The hardware must emit sound alerts using a buzzer when the output of the image processing of the steering wheel is “hands-off”.
- **HFR5** - The hardware must store data packets containing values of speed and alerts (such as hands-off and drowsiness) of at most 4 kb in an external device attached to the embedded system.
- **HFR6** - The hardware must be able to store up to 24 hours of data-packets in an micro SD-Card attached to the embedded system.
- **HFR7** - The hardware must detect vehicle position coordinates using a GPS NEO 6M sensor with frequency of at least 1 update per second and at least 10 meters of precision.
- **HFR8** - The hardware must take at least two pictures of the driver’s face per second.
- **HFR9** - The hardware must take at least one picture of the steering wheel per second.
- **HFR10** - The hardware power supply will be the vehicle’s 12V USB port.
- **HFR11** - The hardware must establish Wifi connection with a smartphone hotspot in order to send data packets.
- **HFR12** - The pictures must be taken using USB infrared cameras to allow visibility in low-light environments.
- **HFR13** - The hardware must use a Raspberry Pi3 microcomputer.

2.3 Software Functional Requirements

- **SFR1** - The embedded software must use neural network and computer vision in the driver’s image to determine the rate of drowsiness of the driver and if driver hands are on the steering wheel.

  SFR1.1 - The embedded software must use the open-source library Drowsiness_Detector to detect driver drowsiness.

  SFR1.2 - The embedded software must use the open-source library OpenCV to detect driver’s hands.
- **SFR2** - The computer vision techniques must rely on the shape of the driver’s mouth and eyes to determine the rate of drowsiness.
- **SFR3** - The embedded software must capture images of the driver’s face using the camera and process them with at least 2 frames per second.
- **SFR4** - The embedded software must capture images of the steering wheel using the camera and process them with at least 1 frame per second.
- **SFR5** - The embedded software must read inputs from the accelerometer and GPS sensor at least once per second.
- **SFR6** - The embedded software must use readings from the GPS sensor to calculate the vehicle speed.
- **SFR7** - The embedded software must create:

  SFR7.1- A hands-off alert whenever the driver hands are not detected in the steering wheel camera processing.

SFR7.2 - A drowsiness alert whenever the rate of drowsiness of the driver is detected over 0.5 in the neural network output.

SFR7.3 - An acceleration alert whenever the acceleration readings are over 2 m/s^2 in any axis (disregarding gravity force).

SFR7.4 - A speed alert whenever the vehicle speed is over 100 km/h.

- **SFR8** - The embedded software must register the speed and the position entry of the vehicle with a frequency of 10 entries per minute.
- **SFR9** - All alerts and speed entries must have a date and timestamp in GMT-3.
- **SFR10** - The embedded software must send all alerts and speed entries to the cloud server when the internet connection is established.
- **SFR11** - The embedded software must buffer all alerts and speed entries in device local memory when the internet connection is not established.
- **SFR12** - The embedded software must send all buffered alerts and speed entries to cloud server after establishing a connection.
- **SFR13** - The embedded software must use JSON format to send data packets to the cloud server.
- **SFR14** - The embedded software must use JSON file to store buffered alerts and vehicle speed.
- **SFR15** - The embedded software must keep a file with last update time of the firmware, to identify when the device is turned off by checking last update time.
- **SFR16** - The embedded software must use Python programming language in the firmware.
- **SFR17** - The embedded software must use Raspbian operational system.
- **SFR18** - The cloud server software must provide a database to store all alerts and speed entries.
- **SFR19** - The server software must be deployed to some free web server host such as Google cloud server.
- **SFR20** - The server software must use Django framework to instantiate database and dashboard pages.
- **SFR21** - The server software must use SQLite database backend.
- **SFR22** - The dashboard software be accessible through a website requiring login/password.
- **SFR23** - The dashboard software must provide CRUD page to register and edit driver profiles.
- **SFR24** - The dashboard software must provide a page to visualize driver profile with history of trips.
- **SFR25** - The dashboard software must provide a page to inspect trips, with trip duration and information about alerts and speed over time.
- **SFR26** - The dashboard software must provide a page to the driver start and finish his current trip.
- **SFR27** - The embedded software must send at least 5 packets per minute

containing acceleration, GPS, Drowsiness and hands-on-steering-wheel data.

2.4 Anti-Requirements

- **AR1** - The drowsiness detection will not work if the driver is using a hat, mask, sunglasses, or any other wearable which blocks the camera from viewing the driver's face.
- **AR2** - The hands-off detection is not guaranteed to work if the driver is using gloves or any other wearable which covers the hands.

3 Development

This section describes the process of the project development, explaining technical details as well as difficulties. Further diagrams can be seen in the project's blog [2].

3.1 Mechanical Structure

Development started with the design and creation of the mechanical diagrams using the OnShape [3], which is an cloud-native product development platform that delivers professional-grade CAD capabilities. The resulting diagram can be seen in Figure 2. The box device was build using MDF(*Medium Density Fiberboard*).

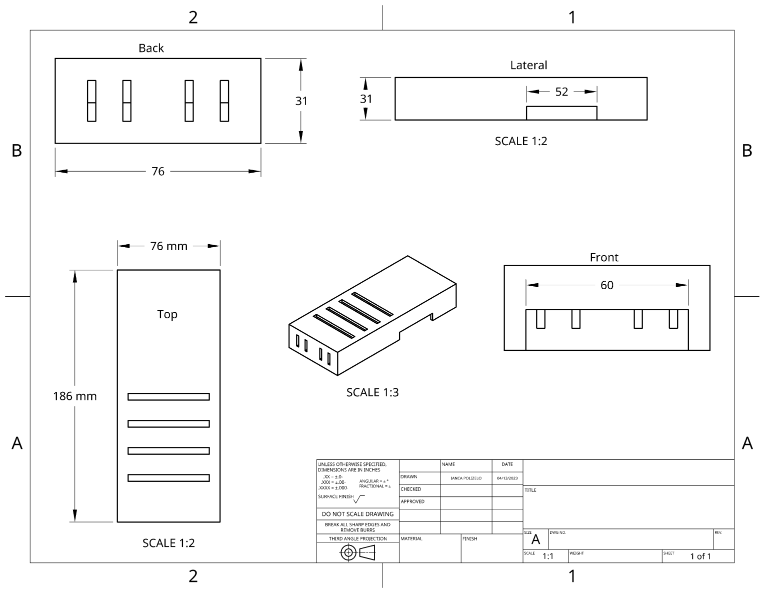


Figure 2: Mechanical Schematics

The main features that the mechanical structure must have are to hold all the physical components present in the hardware safely, but it must be small enough not to block the driver's view, and also protect them from solar radiation, preventing the system from overheating.

The final result can be seen at Figures 3 and 4.

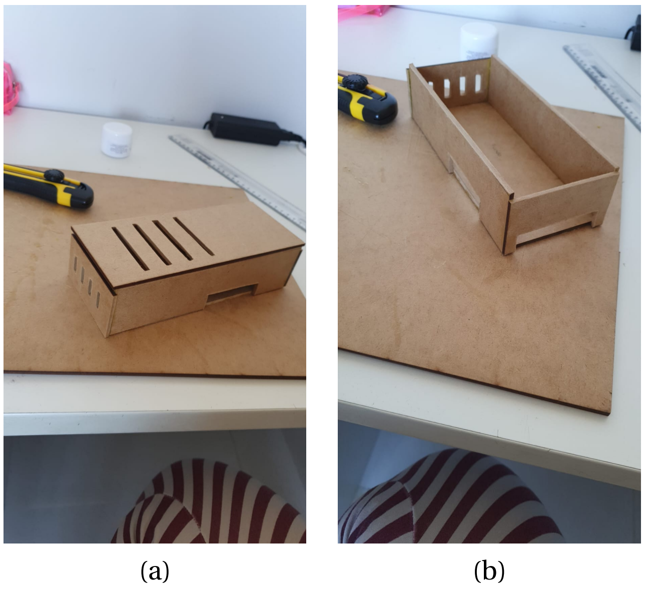


Figure 3: The box device built. (a) Closed box (b) Open box

3.2 Hardware Design

All hardware modules are connected to the microcontroller, as shown in the block diagram in Figure 1. The two system cameras are connected using USB ports, while the buzzer, GPS and accelerometer are connected using GPIO pins and are grouped into a PCB, as shown in the schematic diagram of Figure 5, which was modeled using the EasyEDA tool [4]. Additionally, the system is powered using the vehicle's 12V power supply.

Based on the schematic diagram of Figure 5, a board with the design of the PCB shown in Figure 6 was produced. Figure 7 contains the final result of connecting the boards shown in Figure 5.

3.3 Computer Vision

Getting inspiration from *Drowsiness_Detector* [5], the developed drowsiness detection algorithm makes use of OpenCV library [6] to capture images from the webcam. After capturing the image, a pre-trained facial detector is applied on the image, using dlib [7]. The output landmarks include right and left eye points, which can be used to create bounding boxes for right and left eyes. Then a

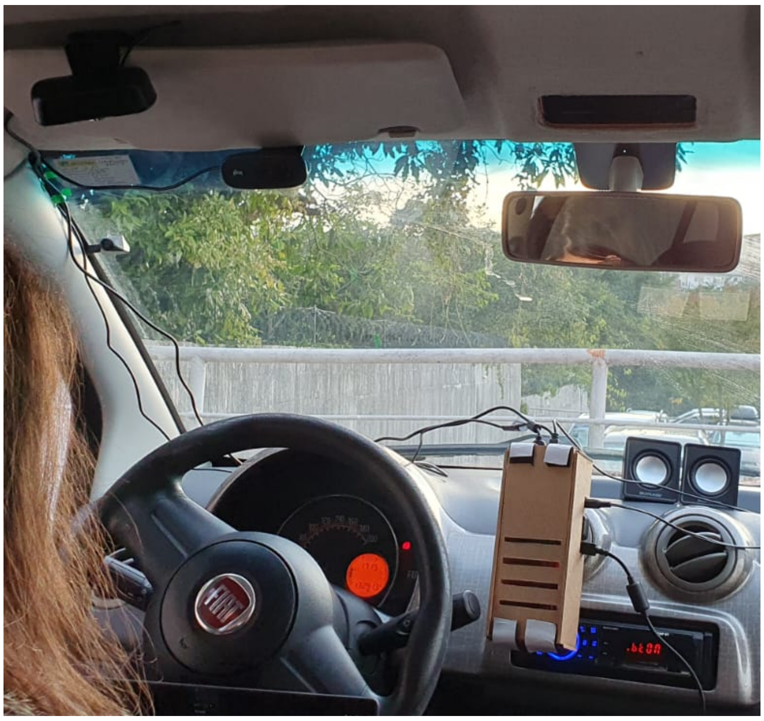


Figure 4: Mechanical structure fixed on the car

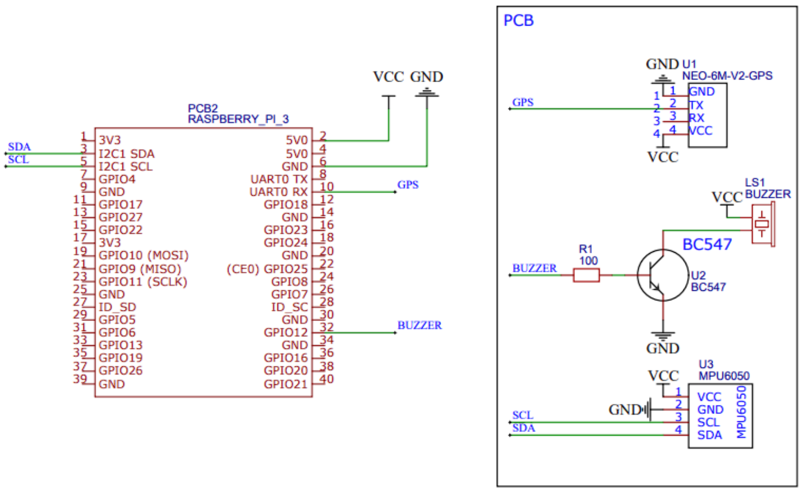


Figure 5: Circuit schematic

heuristic is applied, for the given bounding boxes, the drowsiness level is determined by how small is the aspect ratio of the bounding boxes, meaning that the eyelids are closer together.

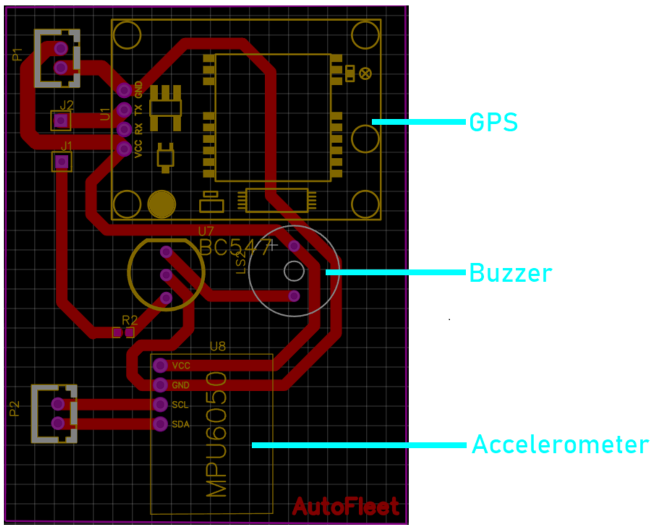


Figure 6: PCB design

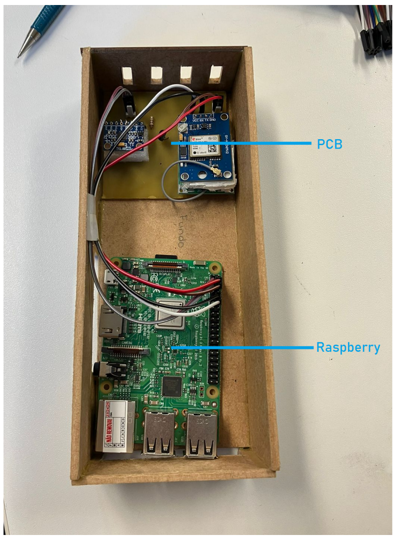
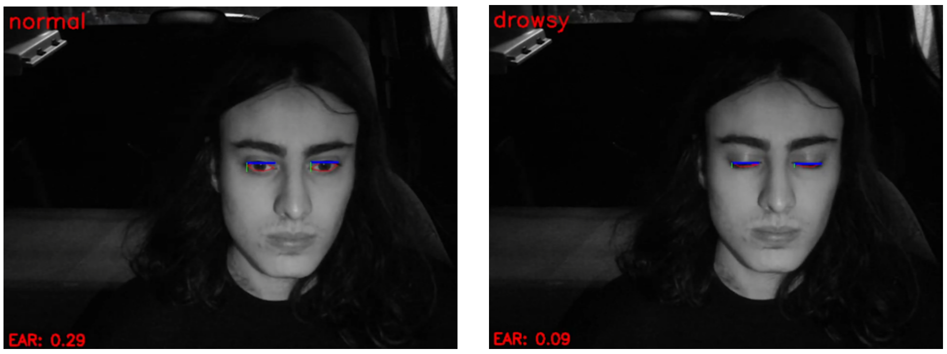


Figure 7: Final result of connection between PCB and Raspberry PI.

This approach depends on the perspective of the camera, so if the person is rotated and not facing the camera straightly, the results will vary. For example, if the driver is looking down, its eyelids become close together when looking from the camera perspective, so it would alert as a drowsiness event. Beyond that, a drowsiness event also depends on a threshold value, which may need to be adjusted depending on the person using the device.

The result of this algorithm can been see in the Figure 8.

For the hands-off detection algorithm, two main components are combined:

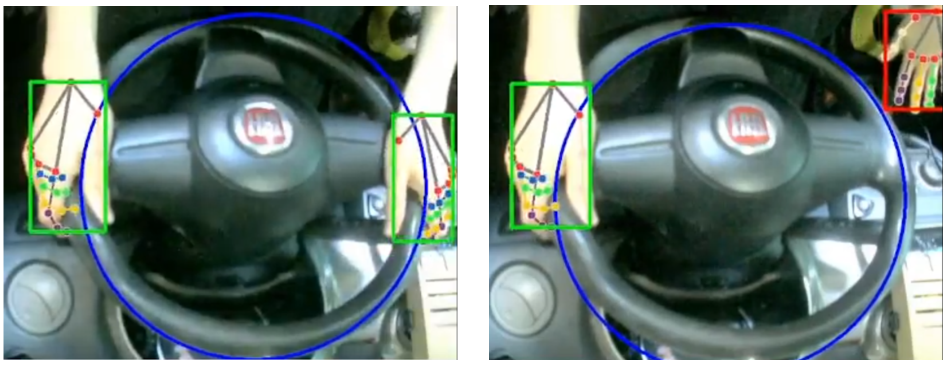


(a) (b)

Figure 8: The bounding boxes of the eyes created in the algorithm. (a) Eyes are open, indicating normal state (b) Eyes are closed, indicating drowsy state

the MediaPipe library [8] to detect hands in the image and the Hough Transform [9] to detect the circle of the steering wheel. Both hands bounding boxes are calculated using the joints points of the hands, then, an overlap test is performed testing the intersection between hands bounding boxes and the steering wheel circle.

The result of this second algorithm can been see in the Figure 9.



(a) (b)

Figure 9: The bounding boxes of the hands and the circle of the steering wheel. (a) Hands touching the steering wheel (b) One hand-off the steering wheel

3.4 Firmware and Backend

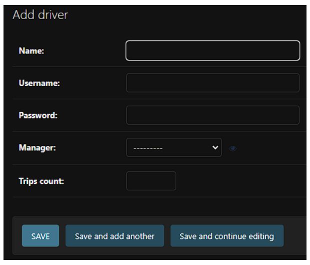
Both the Firmware and Backend are written in Python. Some libraries were used to help create firmware for hand detection [8], drowsiness detection [5], GPS [10], to facilitate I2C communication with the accelerometer [11] and to facilitate pin control Buzzer GPIO [12]. The firmware starts by dispatching different threads for sensors, emitters and the streamer, each of them having different update frequencies.

For the sensors: accelerometer updates once each 2 seconds and GPS updates once each 5 seconds. For the cameras: drowsy detection update twice per second and hands-off detection once per second. Speaker and buzzer updates once each second. Beyond that, the speed measures are derived from the GPS readings by calculating the offset from two consecutive locations and dividing by the time.

Lastly, in a separate thread, there is a streamer that is responsible for fetching data captured with the sensors/cameras and sending them to the webserver. This streamer has an update frequency of once each 15 seconds. For accelerometer and speed data, the streamer only sends the maximum readings, meaning that only the most important of the samples is sent. Additionally, the streamer is responsible for storing packets locally if something wrong happens during the packet transmission, like loss of connection. With that, all the buffered packets are sent to the webserver in batches of 100 packets once the connection is stablished.

3.5 Website

Providing more details about the manager and the driver interaction with the website, a manager has permissions to perform CRUD operations on Trips, Devices and Drivers, as well as being able to analyze data collected from trips (a manager cannot manipulate data collected from trips so that it cannot be corrupted). Figure 10 shows the screens where the manager can perform CRUD operations on driver data, the process is similar for Devices and Trips data.


Add driver

Name:

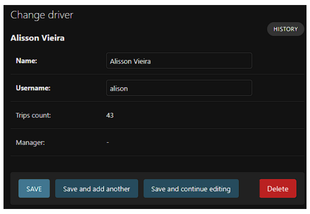
Username:

Password:

Manager: ---------

Trips count:

(a)


Change driver

Alisson Vieira

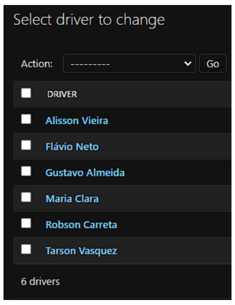
Name: Alisson Vieira

Username: alison

Trips count: 43

Manager: -

(b)


Select driver to change

Action: --------- Go

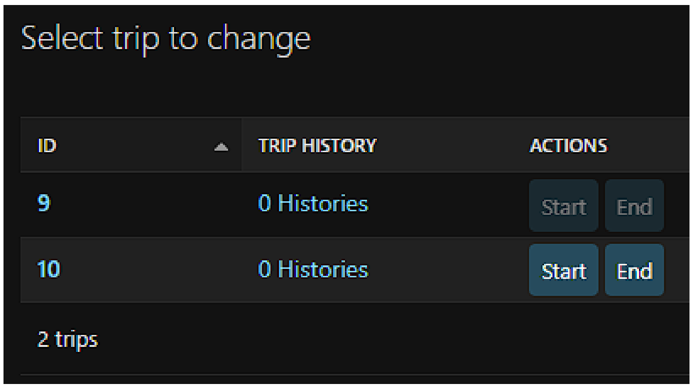
DRIVER
Alisson Vieira
Flávio Neto
Gustavo Almeida
Maria Clara
Robson Carreta
Tarson Vasquez

6 drivers

(c)

Figure 10: Driver creation screen (a), driver modification and deletion screen (b) and driver list screen (c).

On the other hand, a driver can view, start and end his trips as shown in Figure 11. When starting a trip, the trip data packages (named trip history) are collected and stored until the driver ends the trip. A Trip History record contains date and time data, GPS, speed, acceleration, indicators of the driver's drowsiness state and hands off the wheel.



Select trip to change

ID	TRIP HISTORY	ACTIONS
9	0 Histories	Start End
10	0 Histories	Start End

2 trips

Figure 11: Driver trip control screen.

Thus, with the trip packages collected, the manager can inspect the trip, viewing the packages which contains date, time, GPS coordinates, speed (km/h), Acceleration (x, y, z) (m/s^2), absolute 3D acceleration subtracting the gravitational acceleration, hands state and drowsy state as shown in Figure 12. It's important to note that the state is a number which indicates what occurred in that packet as seen in the Table 1.

Table 1: states for hands states and drowsy states.

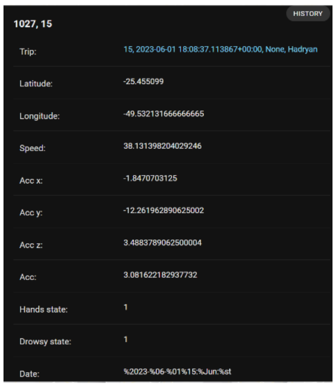
Value	Hands State	Drowsy State
-1	if camera error / initializing	if camera error / initializing
0	if no hand detected on steering wheel	if face not detected
1	if one hand detected on steering wheel	if not drowsy
2	if two hands detected on steering wheel	if drowsy

Finally, it's also possible to check the trip path visually on a map, as shown in Figures 13 and 14.

4 Results

The team was able to develop a prototype that met 98% of planned requirements, not completing the requirement regarding night vision. Despite this single unfulfilled requirement, the team achieved satisfactory results when tried to detect when a driver is falling asleep while driving.

In the end, the prototype is capable of detecting drowsiness status, alerting the driver when necessary, detecting when the hands are touching the steering wheel, storing the data generated during the trip and sending it to the cloud



1027, 15

HISTORY

Trip:	15, 2023-06-01 18:08:37.113867+00:00, None, Hadryan
Latitude:	-25.455099
Longitude:	-49.532131666666665
Speed:	38.131398204029246
Acc x:	-1.8470703125
Acc y:	-12.261962890625002
Acc z:	3.4883789062500004
Acc:	3.081622182937732
Hands state:	1
Drowsy state:	1
Date:	%2023-%06-%01%15:%Jun:%st

Figure 12: Trip history data.

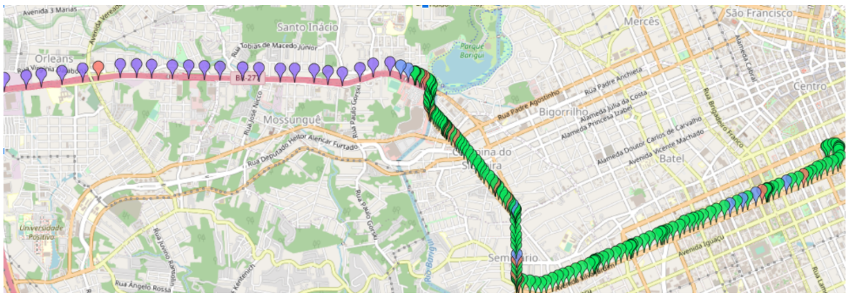


Figure 13: Example of a trip history map showing the driver path.

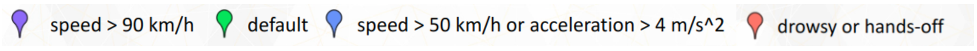


Figure 14: Labels of the driver path pins.

through wifi communication. All this using a small device and a website that works satisfactorily.

4.1 Budget

For the project, it was projected to have a budget about R$200 to R$250 per member of the team, therefore a total budget of R$800 to R$1000. Table 2 shows all components bought and their price, the total cost was R$1096 which is slighty higher than the previewed amount.

4.2 Schedule

The development time for the project was 9 weeks. Table 3 shows an overview of the entire schedule, noting that Estimated Hours include the 30% extra hours.

The full sheet can be viewed at [13].

Table 2: Budget

Amount	Component	Price (R$)
1	Raspberry Pi3 B	586
4	Webcams	210
1	Accelerometer	23
1	GPS	45
1	Buzzer	8
2	Memory Card	78
1	Smartphone support	30
1	MDF Wood	38
1	Wood glue	23
1	Stiletto	25
1	Speaker	30
Total		1096

Table 3: Schedule

Deliverable	Estimated Hours	Hours Worked
Project Charter	19:30	15:20
D1: Blog Site	16:54	15:30
D2: Mechanical Designed	24:42	20:30
D3: Hardware Designed	67:36	44:30
D4: Software Designed	78:00	50:00
D5: Integration	62:24	70:00
D6: Overall Integration	62:24	55:00
TR: Technical Report and Video	48:06	68:00
Total	379:36	338:50

5 Conclusion

The team faced many challenges during this project. At the beginning the hands detection was thought to be a physical sensor in the steering wheel, such as piezo or heartbeat sensor. This approach would be more precise and required less processing. But it is illegal to modify the steering wheel with cables. Instead, computer vision seems a better solution. Using only one camera for this, computer vision was not able to check if the hands were really touching the steering wheel, since this requires a 3D camera. When the hands are above the steering wheel but not touching it, the algorithm also detect as the hands were touching.

Another big problem faced during this project was about night vision camera. Normal webcam have a physical IR filter that blocks it from capturing infrared light. So, the first approach was to remove this filter from 2 webcams, but it did not work. Consequently, one requirement could not be fulfilled.

Despite the many challenges, the team was able to overcome several of them while growing and learning from all the mistakes made at all steps of the development. And at the end, the project was a success with the major components working reasonably well and within expectations.

It‘s important to note that the planning step was essential to this project, especially the risk analysis plan, as without it and the backup plans for it, the project might not have been concluded with success.

References

[1] Portal UOL. Mais de 40% dos acidentes de trânsito acontecem por sonolência, afirma a abrame, 06/09/2019. https://autopapo.uol.com.br/noticia/mais-de-40-dos-acidentes-de-transito-acontece-por-sonolencia-afirma-a-abramet/.

[2] Team AutoFleet. The autofleet blog, 2023. https://www.notion.so/AutoFleet-5ed9cc1781e6449abe93defdd03e8c42.

[3] Onshape. Onshape, 2012. https://www.ptc.com/en/products/onshape.

[4] EasyEDA Team. Easyeda, 2023. https://easyeda.com/about.

[5] garcelling. Traffic monitoring on rpi, 2021. https://github.com/garceling/Traffic-Monitoring-on-RPI.

[6] OpenCV Team. Opencv, 2023. https://opencv.org/about/.

[7] Davis E. King. Dlib, 2023. http://dlib.net/.

[8] Google. Mediapipe, 2023. https://pypi.org/project/mediapipe/.

[9] OpenCV Team. Hough circle transform, 2023. https://docs.opencv.org/3.4/d4/d70/tutorial_hough_circle.html.

[10] Pynmea2 Contributors. pynmea2, 2023. https://github.com/Knio/pynmea2.

[11] Mark M. Hoffman. Smbus, 2023. https://pypi.org/project/smbus/.

[12] Ben Nuttall, Dave Jones, et al. gpiozero, 2023. https://gpiozero.readthedocs.io/en/stable/.

[13] Team AutoFleet. Integration workshop 3 - schedule, 2023. https://docs.google.com/spreadsheets/d/1l-1csUGbnr5T9_vFxL78IYjI2QxsqIY9ozo_I9Vc4Bo/edit#gid=0.