

# FINAL DEGREE PROJECT

SYSTEM TO REDUCE THE RISK OF ACCIDENTS AT STREET INTERSECTIONS  
(ARGOS[1] CAM)



# ARGOS

Name: Ramon Angosto Artigues

Theme: Final Degree Project

Degree: Mechatronical engineering.

Professor: Laura Dempere-Marco

27 October de 2021



## INDEX OF CONTENTS

<b><i>Index of Contents</i></b> .....	<b>3</b>
<b><i>Index of figures</i></b> .....	<b>4</b>
<b><i>Index of tables</i></b> .....	<b>5</b>
<b>1. Abstract</b> .....	<b>6</b>
<b>2. Introduction</b> .....	<b>8</b>
<b>3. Motivation</b> .....	<b>10</b>
<b>4. Objectives</b> .....	<b>10</b>
<b>5. State of art</b> .....	<b>11</b>
<b>6. Methodology</b> .....	<b>15</b>
<b>7. Results</b> .....	<b>34</b>
<b>8. Discussion</b> .....	<b>43</b>
<b>9. Conclusion</b> .....	<b>45</b>
<b>10. Bibliography</b> .....	<b>46</b>
<b>11. Annex</b> .....	<b>51</b>

## INDEX OF FIGURES

Figure 1: <i>Project pipeline</i> .....	9
Figure 2: Distance between the driver and the front of the car .....	9
Figure 3: Obd II wired reader (Image obtained from [25]) .....	16
Figure 4: Raspberry piCamera V2 (Image obtained from [27]) .....	17
Figure 5: Camera position in the car .....	17
Figure 6: Position of the car for image capture .....	18
Figure 7: Examples of image captures in different corners .....	19
Figure 8: Representation of component connections with the Jetson Nano. The materials used for the project are: two Raspberry Pi v2 cameras, used for image capture; a touch screen to view images and interact with the user interface; an OBD reader used for vehicle data collection; a portable battery of the Xiaomi brand, with power capacity of 5v 2.6A; for GPIO connections, four LEDs and a <i>buzzer</i> are used (with the necessary additional components) .....	21
Figure 9: K-NN example obtained from [32].....	24
Figure 10: A) Original image B) NSamples with value 10 C) NSamples with value 3 .....	25
Figure 11: Matrices that make up the Kernels .....	26
Figure 12: Example of the <i>Opening</i> effect (Source: Reference [33]) .....	27
Figure 13: Closing Function Example (Source: Reference [33]) .....	27
Figure 14: Example of the <i>Dilate</i> function (Source: Reference [33]) .....	27
Figure 15: Stages of background subtraction a) Original image b) Blur c) KNN algorithm d) Opening E) Closing F) Dilate G) Find Contours & Bounding Box .....	28
Figure 16: Figurative representation of the separable depth wise layer (Source: Reference [37]) .....	29
Figure 17: MobileNetSSD-v2 Network Structure (Source: Reference [37]) .....	30
Figure 18: GUI screens. A) Charging screen B) Screen while the vehicle is in motion C) Screen with a single camera D) Screen with both cameras .....	31
Figure 19: Electronic circuit generated with KiCad .....	32

Figure 20: Parts of the screen housing, battery, and embedded system A) Embedded system, housing cover B) Base housing for embedded system, C) and D) Rear cover housing for screen E) Screen and battery housing.....	33
Figure 21: Right camera housing. ....	33
Figure 22: Example of the GUI with the car stopped and with the car in motion .....	34
Figure 23: View of the cameras from the driver's point of view .....	35
Figure 24: Comparison of masks of background <i>subtraction</i> algorithms.....	36
Figure 25: Examples of applied background <i>subtraction</i> .....	38
Figure 26: Examples of light warnings in different situations .....	42
Figure 27: ARGOS cam system mounted on a car.....	43

## INDEX OF TABLES

Table 1: Evaluation of <i>Background subtraction</i> algorithms .....	36
Table 2: Comparison of object detection algorithms [35],[11],[22], [40] .....	39
Table 3: Object recognition processing speed in different situations.....	40
Table 4: Sample number by class and TP, FP, and FN values .....	41
Table 5: Results of object recognition evaluation .....	41

## 1. ABSTRACT

### 1.1 ENGLISH

In the following Final Degree Project, the project carried out for the development of a device is exposed, it aims to increase the visibility of vehicles at street intersections to avoid accidents. Specifically, the work will explain the methods used to identify possible hazard sources, the integration of the devices into the vehicle and the driver notification method.

The project has two main methodological lines: 1) acquisition, image processing and object recognition, and 2) design of the electronic device. These two lines of work are integrated into a functional device that, together with the analysis of the vehicle's state of movement, emits sound and luminous warnings indicating the presence of pedestrians or vehicles approaching from the sides.

To carry out this, in the first place, a series of videos have been acquired that have been used for the realization of the tests, prior to the integration into the vehicle. As for the programming, this has been carried out with a Jeston Nano and the Python language together with a series of *open-source libraries*, among which OpenCV stands out. The device's design was carried out with the Fusion360 3D design program.

Therefore, the main objective of the project is to increase the visibility of the driver when he is in a corner trying to turn. The built device allows to increase the visibility of the driver by using two cameras located in the front of the car and will indicate the presence of dangerous objects.

The results obtained are promising. On the one hand, showing the image of the front of the car already represents a great advantage of visibility. In addition, in the videos used to carry out the tests, on sunny or cloudy days and the tests carried out outside the car, satisfactory results have been obtained since the driver was correctly notified when a vehicle or pedestrian was approaching. On the other hand, in adverse conditions, such as rain and fog, some problems of recognition of vehicles and people have been observed. Also, due to the power limitation when connecting the device to a portable battery to incorporate it into the car it has not had the expected operation and has not given the warnings correctly.

In summary, the project has shown results that suggest that this device, with a little more development, could be incorporated as another driving assistance system.

### 1.2 CASTELLANO

En el siguiente Trabajo Fin de Grado se expone el proyecto realizado para el desarrollo de un dispositivo que tiene como objetivo aumentar la visibilidad de los vehículos en las intersecciones con el fin de evitar accidentes. En concreto, el trabajo explicará los métodos utilizados para identificar posibles fuentes de peligro, la integración del dispositivo en el vehículo y el método de notificación al conductor.

El proyecto cuenta con dos líneas metodológicas principales: 1) adquisición, procesamiento de imágenes y reconocimiento de objetos, y 2) diseño del dispositivo electrónico. Estas dos líneas de trabajo se integran en un dispositivo funcional que, una vez procesadas las imágenes, junto con el análisis del estado de movimiento del vehículo, emite avisos sonoros y luminosos indicando la presencia de vehículos o vehículos acercándose por los laterales.

Para ello, en primer lugar, se han adquirido una serie de vídeos que se han utilizado para la realización de las pruebas, previas a la integración en el vehículo. En cuanto a la programación, esta se ha llevado a cabo con un Jeston Nano y el lenguaje Python junto con una serie de librerías *open source*, entre las que destaca OpenCV. El diseño del dispositivo se llevó a cabo con el programa de diseño 3D Fusion360.

Por lo tanto, el objetivo principal del proyecto es aumentar la visibilidad del conductor cuando está en una intersección tratando de girar. El dispositivo construido permite aumentar la visibilidad del conductor mediante el uso de dos cámaras ubicadas en la parte delantera del coche e indicará la presencia de objetos peligrosos.

Los resultados obtenidos son prometedores. Por un lado, mostrar la imagen de la parte frontal del coche ya representa una gran ventaja de visibilidad. Además, en los vídeos utilizados para realizar las pruebas, en días soleados o nublados y en las pruebas realizadas fuera del coche, se han obtenido resultados satisfactorios, ya que se avisa al conductor correctamente cuando se acerca un vehículo o peatón. Por otro lado, en condiciones adversas, como lluvia y niebla, se han observado algunos problemas de reconocimiento de vehículos y persons. Además, debido a la limitación de potencia al conectar la board a una batería portátil para incorporarla al coche no ha tenido el funcionamiento esperado y no ha dado los avisos correctamente.

En definitiva, el proyecto ha mostrado resultados que sugieren que este dispositivo, con un poco más de desarrollo, podría incorporarse como otro sistema de asistencia a la conducción.

## 2. INTRODUCTION

This final project describes the development and prototyping of a system to increase the visibility of drivers at intersections to reduce the number of accidents that occur in these areas.

Reduced visibility increases the risk of collisions with vehicles approaching the perpendicular street to which the driver is going. Especially if the driver has to invade the road, he wants to incorporate, with his car to have visibility. This problem occurs mainly in rural areas where the streets are narrower and at the exits of the parking lots where many times the car has to cross the sidewalk to enter the street, there is a risk of running over a pedestrian.

According to the DGT the number of annual accidents at these intersections is approximately 12,000 on interurban roads and 55,000 on urban roads, this information belongs to group 7 of the DGT, where the number of accidents is specified according to the type of road [2].

The project develops a system to be called ARGOS CAM, referring to Greek mythology specifically to Argos Panoptes[1], who was a guardian ,and servant of Hera, who had many eyes, which made him very efficient for surveillance because he always had one open to watch. The project logo is a reference to this myth.

ARGOS Cam consists of two main parts: one that is responsible for the collection of vehicle data (i.e. speed) and a second part that is responsible for capturing images. This second module - focused on image processing - is responsible for offering help to the driver, through image recognition, and is modulated by the first, which focuses more on aspects of vehicle electronics. The structure of ARGOS Cam is presented in Figure 1, which includes the structure of the project:



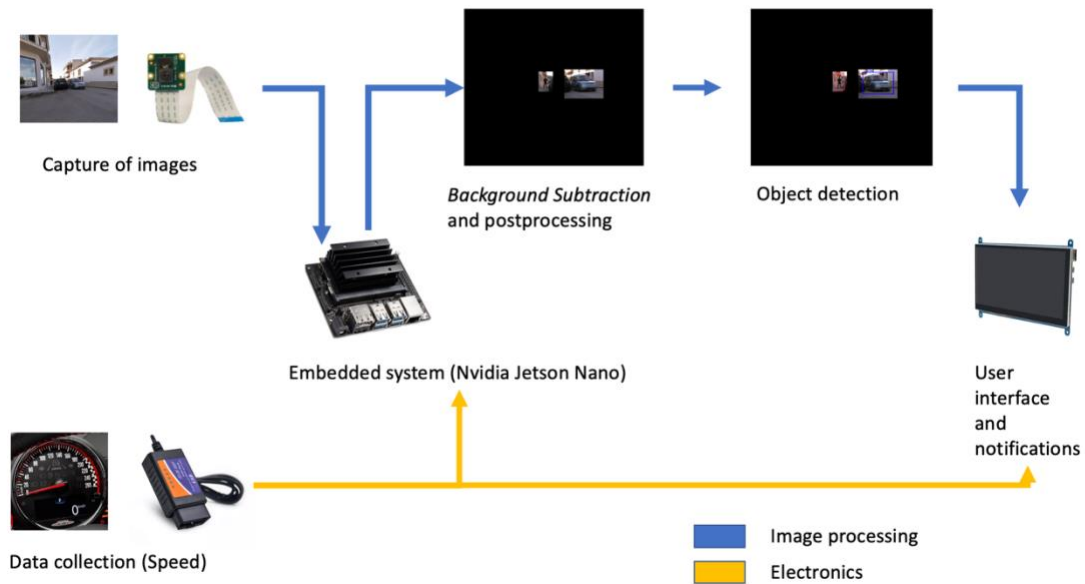


Figure 1 Project pipeline

In Figure 2 you can see an image that represents the advantage of the system for the driver, if two cameras are placed at the front of the car.

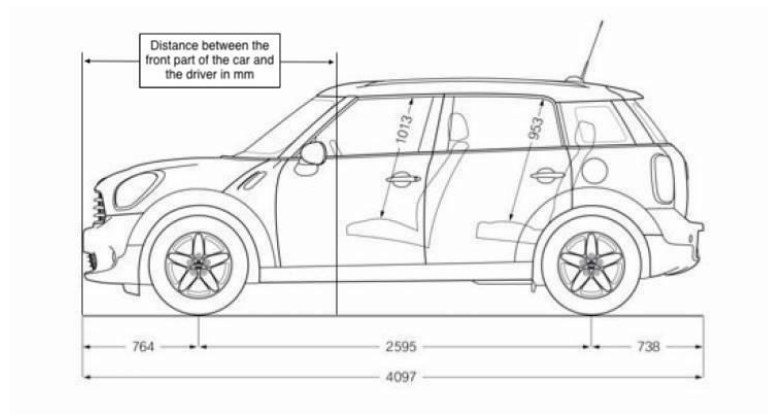


Figure 2: Distance between the driver and the front of the car

It should be said that, in addition to the intrinsic visual aid that the cameras will provide, image processing algorithms will be used to warn the driver if any element appears to which attention has to be paid, such as a car that is approaching or a pedestrian that is crossing the street.

### 3. MOTIVATION

The idea of the project starts from the identification of a problem that I have been able to see driving, and as already been mentioned in the introduction, it is difficult to see if other vehicles are approaching the intersections. It is true that in some corners there are mirrors that aim to solve these problems, but they are not always there and are usually in poor condition, which limits their use.

After identifying the problem, I have conducted research to see the number of accidents at these intersections, and the alarming number of 67,000 accidents a year of this type has become the other reason that has motivated me to carry out this project, to try to reduce this number of accidents and make people's lives easier.

In addition to trying to make life easier for people, working with *Deep Learnig* algorithms is a challenge that stimulates me and allows me to apply the knowledge acquired throughout my studies in mechatronics engineering and offers me the opportunity to deepen my knowledge that can be useful for a future workplace in the programming environment.

### 4. OBJECTIVES

As mentioned in the motivation of this project, the overall goal is to reduce the number of accidents that occur at intersections due to poor visibility.

Specifically, the development of a device that can be integrated into the car, which allows the visualization of the street to which the driver wants to join without having to invade it and that the driver is notified if another vehicle approaches or there is a pedestrian crossing or with the intention of doing so.

The project also has two specific objectives. First, the collection of images and data obtained from the vehicle, along with artificial intelligence algorithms to help the driver on such roads. Secondly, the design of a graphical interface and an alert system to incorporate into the vehicle.

The objectives examined can be sorted in order of priority as follows:

1. Reduce the number of accidents on this type of road because of poor visibility
2. Make use of cameras and information provided by the car's sensors, along with artificial intelligence algorithms to aid in driving.

3. Design a device with an alert system for incorporation into the vehicle

## 5. STATE OF ART

At this point, the current state of the literature that exists on the use of driver assistance devices and each of the points mentioned in the introduction that make up the project will be discussed.

First, advanced driver assistance systems, also known as ADAS[3], have been in place in the automotive industry since 1950, making driving safer. Some examples of these early uses were speeding sensors and the ABS braking system. These systems are now integrated into all vehicles, and the European Commission proposes to make it compulsory to incorporate more assistance systems to reduce the number of accidents.

Today, many more have emerged from these systems that help to prevent accidents. In addition to roadside assistance, they are also related to driver assistance within cities. Automatic proximity braking systems, with ultrasound sensors and LIDARS (Light Detection And Ranging<sup>1</sup>) and parking assistance systems - with cameras and proximity sensors - are two of the clearest examples in this use case. The Nissan company presented in 2007 a concept of car that had incorporated 4 cameras around the car that allowed to simulate a drone view of it to assist during parking, in addition to allowing to select the cameras that were seen on the screen [4]. It also incorporated ultrasound systems to alert nearby objects.

Autonomous cars work mainly with lidars, cameras or the combination of these two technologies (or similar)[5]. This combination is known as a sensor fusion and allows to avoid some of the problems presented by each technology.

In this work we have opted for an approach like that adopted by Nissan and used by autonomous cars that use cameras and artificial intelligence systems.

### 5.1 VEHICLE DATA COLLECTION

---

<sup>1</sup> LIDAR: Sensor that allows to determine the distance to an object or surface by means of a pulsating laser. [16]

As for the collection of vehicle data, in 1989 the OBD I port was introduced to American cars, which allowed the control of the state of the engine through error codes. And in 1996 the OBD II port is presented, which in addition to allowing the visualization of error codes allows the reading of vehicle data such as speed, revolutions and other parameters not related to the engine. Since 2003 this port is mandatory on all vehicles. [6, 7]

## 5.2 IMAGE CAPTURE

Another key aspect of the developed system is image capture. In this sense, there are two main types of connections, used in embedded systems for autonomous driving work and for integration into embedded systems. These two types are: MIPI CSI-2 and USB 3.0

On the one hand, the CSI-2 MIPI connection is the most used in embedded systems and mobile telephony. This allows for faster image transmission, 6 GB/s of bandwidth, and more efficient than USB 3.0, as it does not take up part of the CPU to operate. They also tend to have a smaller size of both the camera body and the cable. On the contrary, it presents some disadvantages such as the need for special libraries and the difficulty of using different sensors in the cameras [8].

On the other hand, the USB 3.0 connection reaches up to a band width of 5 Gb/s and only needs to be connected to work. Because the type of connection is more common, it is easier to find a compatible device or change a broken device or improve the system [8], [9].

In addition to the connection there are other parameters that influence the capture of images, that is if the camera is monochrome or with a color sensor, if it is equipped with a full or rolling sensor shutter release, if it has autofocus or not, if it is equipped with night vision capabilities, as well as other parameters that are not relevant to this project [9].

Since this project will use an embedded system, taking into account what has been discussed in this section, it has been chosen a camera with the CSI-2 MIPI connection, color sensor without night vision or autofocus - since a very fast movement can occur in front of the camera and the self-focus could be slower than the movement it produces.

## 5.3 EMBEDDED SYSTEM

Embedded systems are special-purpose computer systems designed to perform one or a few dedicated tasks, and generally limited to real-time computing. It is usually integrated as part of a complete device, being the entire device processing unit. These embedded systems are found in most consumer devices currently in use [10].

Today there is a wide variety of embedded systems on the market, but in most prototype projects at reduced cost the decision comes down to two: the Jetson Nano [11] and the Raspberry [12]. These two boards can run an operating system and act as a small capacity computer.

On the one hand, the Raspberry was created in 2012 to teach programming and how to interact with electronic devices. It has a RAM that ranges from 1 GB to 8 GB, which allows you to run parallel tasks, and a 4-core CPU. On the other hand, the Jetson Nano is a board designed by the company NVIDIA, designed for the execution of *machine learning* and image processing programs. It has two versions. A 2 GB of RAM and one of 4 GB, the 4 has the advantage that has two CSI MIPI ports, to connect two cameras in parallel. It also has a GPU, this provides a great capacity for parallel calculation of complex tasks such as image processing and *Machine Learning*.

Because the project requires two cameras and is primarily based on image processing, the integrated Jetson Nano system will be used.

## 5.4 MACHINE VISION ALGORITHMS FOR ASSISTANCE AT STREET INTERSECTIONS

This section reviews the main computer vision algorithms that have been developed to solve the tasks that concern us in this project: *Background Subtraction* and object detection.

### 5.4.1 BACKGROUND SUBTRACTION

*Background Subtraction* is a technique that consists of separating the *background* from those objects that are in motion in a video. One of the functions in which it is most used in the video surveillance, since it allows a more simplified detection of the objects that are in motion. *The Background Subtraction* consists of a series of stages. These are: 1) the initialization of the background (where a first image is normally taken), 2) the initialization of the model (which will be used for the definition of what belongs to the background and what is in motion), 3) the mechanism for updating the background, and 4) the classification of the images that are entered [13].

Traditionally, the most commonly used methods for this task are *the Gaussian Mixture Models (MoG)*[13], as these methods take into account changes in lighting, image noise, and slower-moving objects.

In addition to this there are also other models for background detection that have appeared in recent years. Some of them rely on algorithms to segment the background through neural networks and semantic segmentation, which maps each pixel in the image to a class [12, 13]. There are also researchers who have focused on the use of other techniques based on *Machine Learning*, such as *K-Nearest Neighbours*[13].

Before the final decision to use the algorithm based on K-Neares Neighbour (K-NN), a comparison of the different algorithms has been made. In this paper, due to the limitations in the extension, we do not show an exhaustive analysis of all the results obtained, but a conclusion was drawn from this study and it is presented in the discussion section.

---

#### 5.4.2 OBJECT DETECTION

Object detection consists of combining two tasks, classifying an object that appears in the image and locating it, that is, saying where one or more objects are in the image [16]. In the field of computer vision, different methods have been developed to carry out this task, both from the perspective of classical computer vision and that based on Deep Learning. Since in this project we will focus on this second approach, we review in this section the main methods developed in the field of object detection.

However, it is necessary to first establish some concepts and introduce what *Deep Learning* is, as well as introduce the notion of Transfer Learning. *Deep Learning* consists of the use of organized structures that emulate the processing of information that takes place in the nervous system. This structure has data processing units, called neurons connected to each other through *synapses* characterized by weights that the algorithm learns from the data. Image processing is an area that has benefited greatly from these data processing techniques. The company NVIDIA has contributed substantially with the creation of the CUDNN library, which allows the use of the computer's GPU to speed up this process [17]. This method of approaching the nervous system allows to imitate it by creating specific areas to carry out specific tasks, such as extracting features from an image [18].

On the other *hand*, *Transfer Learning* consists of using the knowledge acquired in previous tasks to train a new task [19]. This is a similar way of transmitting knowledge to what humans have, compared to the traditional learning methods found in classical machine *learning*

approaches. For example, if a neural network has been trained in the first task to recognize plants, and later we want to train a different network to recognize trees, some of the knowledge of the first network can be reused for the second, with less information needed to train the second.

To carry out this task there are several *Deep Learning* algorithms. Here are some of them:

- **R-CNN models:** The model was introduced in 2014 by Ross Girshick and is based on convolutional neural networks. The network is divided into three stages: it proposes a region of interest, the characteristics of the same are extracted and finally classified [20].
- **Fast R-CNN models:** they are based on the previous model and were improved by the research group of Microsoft, unlike the previous one, it simplifies the proposal stage of regions of interest and combines the last two stages, allowing to accelerate the recognition process [21].
- **YOLO (You only look once):** this is a network that has as input an image, and results in a bounding box and a classification. This technique is less accurate than Fast R-CNN models but offers higher processing speed. The operation of this algorithm is based on the division of the image, in grids, and different *bounding boxes* are proposed, these will be determined later if they belong to a class based on the trust assigned to them [22].
- **Single Shot Detector (SSD) methods:** They are designed for real-time object detection. They are unable to match the accuracy of Fast R-CNN networks, with a low-quality image. They achieve increased processing speed by eliminating the proposed regions and increase accuracy by using feature extraction at multiple scales. This method consists of two parts a feature extractor based on a neural network and a series of convolutional filters for object detection [23, 24].

In the context of this project, object detection is the most important part, as it is responsible for giving the driver the warning of what is happening. The methodology section will explain the selected model.

## 6. METHODOLOGY

### 6.1 VEHICLE DATA COLLECTION

The only data collected from the vehicle will be the speed. This is used to turn the screen on and off so that it does not distract the driver while on the move. 2 km/h has been selected as the point at which the display is activated. That is, the screen will be active as long as the vehicle is moving at a speed of less than 2 km/h or is stopped. The small margin to zero is given so that the screen does not activate and deactivate every time the vehicle advances a few centimeters if it stops behind another car waiting for it to be its turn to turn or advance.

As already mentioned in the state of the art, the method used for data collection will be the OBD II port. Before the decision to use the OBDport, other alternatives have been proposed, such as the use of GPS or the direct reading of the tachometer through a cable. The first option has been ruled out due to the lag between the GPS reading and the speed of the car, especially at low speeds. The second method has been ruled out so as not to damage the car to be used for testing, despite being the most reliable method.

The OBD port works as follows: at the bottom of the steering wheel is usually the connection point. This collects data from all the sensors in the car and constantly monitors them. Once the reader is connected, which can be Bluetooth, cable or WIFI, the vehicle data is transmitted to the device used for reading data. This project will use a reader with USB cable communication, such as the one in the image below.



**Figure 3: OBD II wired reader (Image obtained from [25])**

The data transmitted from the sensor is in the form of a 5-digit code. The first digit indicates which part of the car the reading is about, the second the organization that has defined the code, the third specifies a specific function of the car and the last two are to indicate errors [7].

In order to simplify the reading of these codes, the Python-OBD library[26] will be used, which has incorporated functions that allow the codes to be read and their values to be provided directly. Once the reader is connected, the connection is established and a function is declared to monitor the desired value, in this case the speed.



## 6.2 IMAGE CAPTURE

Any machine vision system has a first phase of image acquisition. For this task will be used two cameras Raspberry piCameras V2 of 8 Megapixels. This are placed at the front of the car, in image 5 you can see the point where they are placed.

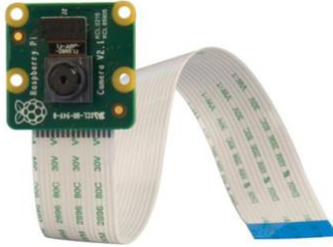


Figure 4: Raspberry piCamera V2 (Image obtained from [27])



Figure 5: Camera position in the car

To obtain the images of the cameras, the OpenCV library will be used, of which more details will be given later, and the GStreamer multimedia *framework* that allows you to adjust the output format of the image, adjust the dimensions of the captured image and the capture frequency (framerate).

*Framerate* is a very important parameter in this project as the system must be able to capture moving cars. In principle the urban speed limit is 30 km/h, but as a margin of safety it will be put that up to 60 km/h (17.7 ms/s) must be detected in case a vehicle goes at a speed higher than the allowed speed.

This speed makes it possible to determine the minimum necessary to be able to capture cars at 60 km/h. In this way, if 18 images per second had been taken, the car would have moved approximately 1 meter. As the *framerate* increases the distance that the car moves each

frame, it is reduced. In our case the *framerate*  $f$  is set to 60 FPS, <sup>2</sup> in order to allow a maximum movement of the car between captures of 30 cm.

Therefore, the camera is adjusted with the following parameters:

Capture frame size: 1280 x 720-pixel pixels

*Framerate*: 60 FPS

It is worth anticipating that the 60 FPS will not end up giving as the algorithms explained below will make the capture process slower.

Figure 6 shows a representation of the position from which images are captured in the project. As you can see, the driver does not invade the perpendicular track.

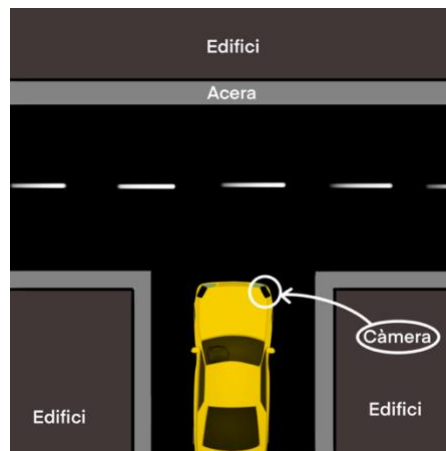


Figure 6 Position of the car for image capture

Below are a series of images taken from the indicated position, and further examples can be found in Annex 1.

---

<sup>2</sup> Fps: *Frames per second*



Figure 7: Examples of image captures in different corners

### 6.2.1 ETHICS OR PROTECTION OF IMAGE RIGHTS

Since this project captures images, it is necessary to clarify that none of the images that the device uses for processing is stored, thus maintaining the privacy of people who may appear at the time the cameras are working.

The only time the images have been stored has been when a sample data set was prepared to carry out the tests, and in these, only the people who have given their consent to appear in the videos appear.

## 6.3 EMBEDDED SYSTEM

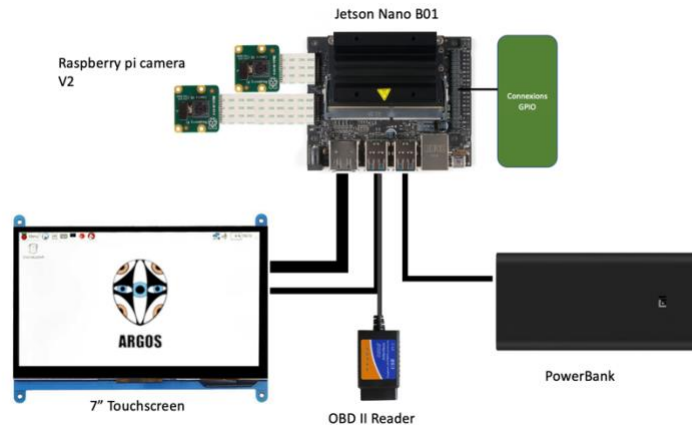
As mentioned above, the integrated system that will be used in the project is a Jetson Nano B01 from the Nvidia brand. This board is the place where all the data processing will be given. The operating system you have installed is Jetpack 4.5, which is based on Linux 18.0. The programming language that will be used to perform data processing and driver alerts will be Python 3.7.

The program that controls the project makes use of the following libraries:

- **Python-OBD:** As mentioned in the section of reading vehicle data this library, allows communication with the OBD reader, and allows you to obtain the speed of the car to be able to use it and regulate whether the screen is on or not.
- **OpenCV (with CUDA):** It is the library that takes care of all the image processing and the capture of the video. It is within this, that are Background Subtraction which are explained in the next section. In addition, this library together with the Tensorflow library is responsible for object detection. Because the integrated system has a GPU that belongs to Nvidia, you can benefit from the CUDA library, which offers an accelerated image ado process.
- **Tensorflow:** is an open-source library developed by Google, which is used for the training and execution of Deep Learning algorithms.
- **Jetson inference:** the function of this library is the optimization of neural networks so that they can be used in the Jetson Nano.
- **PySimpleGUI:** This library is used to create the user interface so that it interacts with the screen and can see the cameras.
- **Jetson.GPIO:** allows you to control the GPIO pins of the Jetson, allowing you to activate the LEDs and sound warnings that will be used to notify the user of the presence of a recognized object.

The code written for the project is included in the attachments, it consists of two *scripts*. The first includes a class that was created to manage image processing algorithms. Includes *background subtraction and image* recognition algorithms. The second script, serves to encompass all the code, and allows the reading of the OBD. It also manages the GUI and communicates with the first *script* to get the results of image processing.

The following figure shows the scheme of connections, to understand how each element communicates with the Jeston Nano.



**Figure 8: Representation of component connections with the Jetson Nano. The materials used for the project are: two Raspberry Pi v2 cameras, used for image capture; a touch screen to view images and interact with the user interface; an OBD reader used for vehicle data collection; a portable battery of the Xiaomi brand, with power capacity of 5v 2.6A; for GPIO connections, four LEDs and a buzzer are used (with the necessary additional components)**

As you can see in Figure 8, the board is powered through a power bank. This is not the most recommended option, and it would be better to power the plate directly with the car battery and a 12 V to 5 V reducing converter, but due to a problem with the car battery used for testing, this solution has been chosen.

## 6.4 BACKGROUND SUBTRACTION

Images acquired by front-facing cameras are processed through the *Background Subtraction algorithm* to isolate those objects that are moving in the scene. Below are the criteria used to select the algorithm to work with and the details of how it works.

### 6.4.1 SELECTION OF ALGORITHM

To select the most suitable algorithm for this project, different *Background Subtraction options* have been evaluated. The following algorithms have been considered: k-NN, MOG, MOG2 and GMG.<sup>3</sup> To do this, these algorithms have been

<sup>3</sup> GMG: Algorithm is based in the article [42]. The basis of its operation is the estimate of image and the app per pixel and Bayesian filters.



applied in a series of videos, and the resulting masks have been saved. These have been compared to the expected result.

These expected results have been obtained by manual tagging with the Photoshop graphic design tool. With this, a representative *ground truth* mask has been created, following the form of labeling used by the algorithm, in other words, a *bounding box* has been made around the moving object as the algorithm should do. The color black for the background, with a label of 0, and the color white for the moving element, with the label 1.

The algorithm used for this evaluation was obtained from the following references [28, 29]. The performance of each of the algorithms has been evaluated by a confusion matrix, which represents the four possible combinations between the real values and those obtained from the binary mask generated by the algorithm.

In our case, the background class belongs to the negative value, that is, 0, and the foreground class or moving object would be the positive value, that is, 1.

The confusion matrix is defined from the following classification results:

#### MATRIU OF CONFUSION:

---

**True positive (TP):** Represents the number of positive predictions, that is, the number of pixels that have been labeled with class 1 and that belong to this class.

**True Negative (TN):** Represents the number of negative predictions, that is, the number of pixels that have been labeled as class 0 and that belong to that class.

**False positive (FP):** Represents the number of samples that have been classified as positive (foreground) but are negative (background).

**False negative (FN):** Represents the number of samples that have been classified as negative (lower) but are positive (foreground).

#### ALGORITHM PERFORMANCE MEASURES:

---

From this, the following measures are obtained:

**Accuracy:** This is the correctly classified pixel aspect ratio compared to all evaluated pixels. This measure is very sensitive to the evaluation of data sets in which the classes are not balanced. Equation 1 shows how it is calculated.

$$Precisió = \frac{TP+TN}{TP+TN+FP+FN} \quad (\text{Eq. 1})$$

**Sensitivity or ratio of true positives:** Corresponds to the proportion of samples classified as true positives (TP), with respect to all positives (TP + FP). Equation 2 is defined as making this measurement. This measurement can be interpreted in how well the algorithm detects objects that are in motion.

$$Sensitivitat = \frac{TP}{TP+FP} \quad (\text{Eq. 2})$$

**Specificity or false positive ratio:** On the other hand, what is specified is the proportion of negative samples classified as positive (FP) with respect to all those that are negative (TN + FP) as shown in equation 3. This value is interpreted as the ability to correctly detect the background of the images obtained.

$$Especificitat = \frac{FP}{TN+FP} \quad (\text{Eq. 3})$$

**F1 value:** This is a measure that relates precision values and sensitivity. This measurement is quite useful when you have a great inequality in the data used for the evaluation, and this is one of the cases in which it is necessary to use it, since most of the pixels of the images belong to the background. The following equation shows how the calculation would be performed.

$$F1 = 2 * \frac{(Accuracy*Sensitivity)}{(Accuracy+Sensitivity)} \quad (\text{Eq. 4})$$

In the results section, the results obtained in this study will be presented. These results and their implications are discussed in the discussion section.

#### 6.4.2 KNN BACKGROUND SUBTRACTION

The selected algorithm is based on the *Gaussian Mixture Model* (GMM), and el *K-nearest neighbor* (K-NN). This is described in depth in [30]. Generally speaking, a recursive method is presented to update the GMM parameters and update them for each pixel. The K-NN

algorithm is included in the OpenCv library [31] and has a number of parameters that must be adjusted to achieve the desired operation of the algorithm. Model initialization is performed with the number of *frames* assigned to the NSamples variable. During this period, you define which objects belong to the background class and which objects of the moving object class by comparing the pixels in these frames. The model is then updated every *N* of *frames*, where *N* is the number of frames that have been defined in the History and parameter.

K-NN is an algorithm used primarily for predictive classification problems. This algorithm does not have a specific training phase, but during the same training a classification task is carried out. It is a non-parametrical algorithm, i.e. the training is carried out without information about the input data [32]. The operating principle of the algorithm is based on a voting system carried out between the K nearest neighbors and the similarity of characteristics - with respect to the training data - to predict the classification of the new data. The following image presents a simple example of how the algorithm works. The black dot is the new information being evaluated. If K=3, we have to look at the three points that are closest. Because most of these are red, the new point is assigned to the red class.

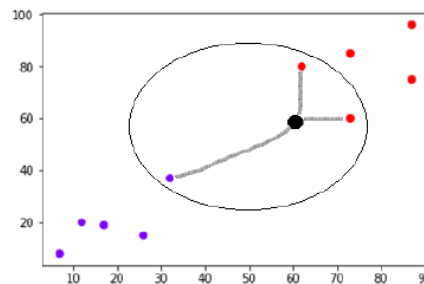


Figure 9: K-NN example obtained from [32].

The parameters needed to adjust the operation of the algorithm are:

**History:** This is the number of *frames* that will affect the modification of the parameters assigned to the background class and the moving object. With this parameter you can adjust the reference on which the new *frame* will be compared to check if it has changed or not. This value has been set to a value of 100. If the value is too small the changes in the image take too long to appear reflected, and if the value is too high, the mask will change too often causing the minimal change in lighting to drastically modify the mask. The *framerate* of the camera will affect the value of this setting, since the higher this parameter, the less time it takes to update.



**Nsamples:** Defines the number of samples saved in memory with which the *Background Subtraction* algorithm must compare and assign a new class to the pixels of the new *frame*. If you have a very small number of samples, the background segmentation has a lot of noise as you can see in the image below. On the other hand, with the adjusted value a more uniform mask is achieved. A value of 10, has been used for this project



Figure 10: A) Original image B) NSamples with value 10 C) NSamples with value 3

**KNNSamples:** The parameter K-NN Samples, is the variable K that is always defined in the K-NN algorithms. With this, the number of samples with which each pixel will be compared is adjusted to decide if it belongs to one class or another, the one with the most votes will be assigned. In the original article for this algorithm, we recommend that you use a value that is between 3 and 60. [30] As the K-value increases the accuracy with which the background is detected it also increases because the current pixel is compared to a larger number of nearby pixels to be classified as background or as a moving object, but as a negative effect of this increase the speed of acquisition of the mask is reduced. In this case a value of 3 will be used, because a higher value makes the algorithm work slowly.

**Detect shadows:** By modifying the value of this variable you can select whether the shadows will be considered as background or as moving objects, depending on whether the variable is active or not, you can adjust it to discard only marked shadows or all those that appear in the *frame*. For this project, all shadow types will be detected because any small change in the image can be used to obtain faster recognition of the object.

Once the *Background Subtraction algorithm* has been applied, a series of morphological operations will be performed on the binary image obtained in order to facilitate the detection of objects.

### 6.4.3 MORPHOLOGICAL TRANSFORMATIONS

Morphological transformations are operations based on the geometric structure of the objects in the image. These have as input an image, usually binary, and a structural element or *kernel*. Using the combination of these two and the desired operation, the mask is modified[33]. In this work two different *Kernels* are used, which allows their application in different cases. The first is crosshair shaped and 5x5 pixels in size, and the second is ellipse-shaped, with a pixel size of 7x7. The elliptical shape covers more surface area, allowing the mask to expand more significantly.



Figure 11 Matrices that make up the Kernels

The morphological transformations used are carried out in the following order:

#### OPENING

First, it is to carry out an **Opening**, this operation is the combination of two operations, an erosion followed by a dilation of the mask. Frequently used to remove noise in masks, so it has been used in the first place. In this way, small elements that may have been detected erroneously are removed. This operation will be the only one that will use the cross-shaped *kernel* to prevent the rest of the mask from being too modified.



Figure 12: Example of the *Opening* effect (Source: Reference [33])

#### CLOSING

---

This is the operation contrary to the **Opening**. In this case its function is to close the small openings that may remain in the mask.



Figure 13: Closing Function Example (Source: Reference [33])

#### DILATE

---

Finally, using the **Dilate** function, the mask will expand to reduce the chance that some part of the moving object will remain unmasked.



Figure 14: Example of the *Dilate* function (Source: Reference [33])

#### 6.4.4 CONTOUR SELECTION AND BOUNDING BOX

To finish the *Background Subtraction*, the contours of the mask obtained after the morphological transformation will be searched, since it may be that in the previous processing there could be openings in the mask. These contours obtained will be used to

generate Bounding Box -which will encapsulate those contours that have passed a size filter- achieving the objective of expanding the masks of moving objects.

The following illustration shows the different steps that the algorithm follows to isolate moving elements from images. The imagen A is the original. Then this is blurred (to remove noise), and the **background subtraction** algorithm is applied. In image D you can see how the small dots of the car in image C disappear when the **opening** has been applied. In images E and F, you can see the effect of applying closing and **dilate**. Finally, the bounding and bounding box algorithm creates a rectangle around the moving object.

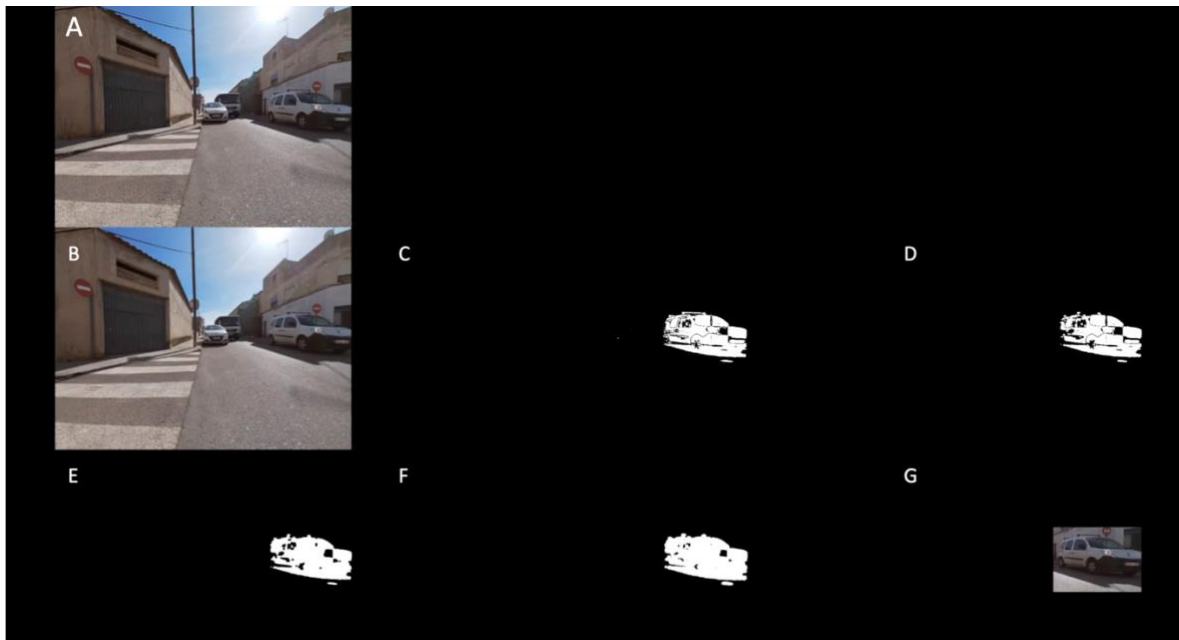


Figure 15 Contours & Bounding Box

## 6.5 OBJECT DETECTION

After the *Background Subtraction* has been applied, object detection is performed. Thanks to the previous step, static objects, such as parked cars, will not be recognized. This prevents the driver from being misreported, who must be alert to objects that may pose a hazard (for example, an approaching car or a pedestrian crossing the street).

To perform this task, the Mobilenet-SSD v2 network will be used pre-trained with the COCO(Comon Object in Context) data set [34], being able to recognize up to 90 different

objects and the background [35]. This network consists of two main components: MobileNet [36], a deep neural network with an efficient architecture designed to be used in devices with less computing capacity such as phones and embedded systems, and an SSD (Single Shot *Multibox Detector*) network [23], which allows the simultaneous detection of multiple objects in an image.

## MOBILENET

It is a convolutional neural network that works as a classifier. The feature that sets it apart from the rest - and allows it to work more efficiently on lower-performance systems - is the use of a new type of convolutional layers called *Depth wise Separable Convolutions*. These are based on a normal convolutional layer, of the desired dimension, but in this case two 1x1 convolutional layers are added, one in front and the other at the end. The first has the function of expanding the tensor and the last one that acts as a bottleneck reducing the number of tensors. In other words, it is as if the data collected in the image were decompressed and once processed the result was compressed [32,33].

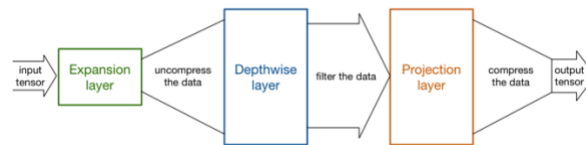
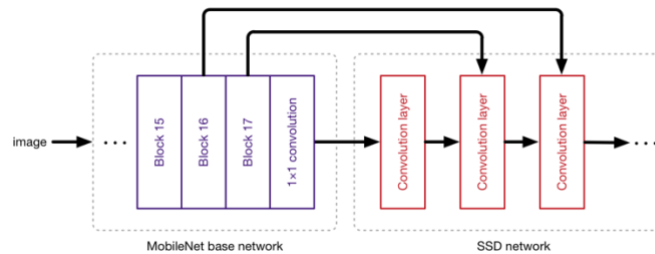


Figure 16: Figurative representation of the separable depth wise layer (Source: Reference [37])

## SSD

Performs sorting and localization in a single pass through the network while predicting the class and *bounding box* as it processes the image. Image 17 shows the architecture you have. In this you can see that Mobilenet is integrated into the first part and works as a feature extractor for SSD. This is possible thanks to the flexibility of the SSD that allows to change the basic architecture that [37] uses.

This network generates a fixed number of *bounding boxes* and a score for the presence of instances of objects of a class within these *bounding boxes*. Finally, it ends with a *Non-maximum suppression* step to group bounding boxes with a high degree of overlap. [23]



**Figure 17: MobileNetSSD-v2 Network Structure (Source: Reference [37])**

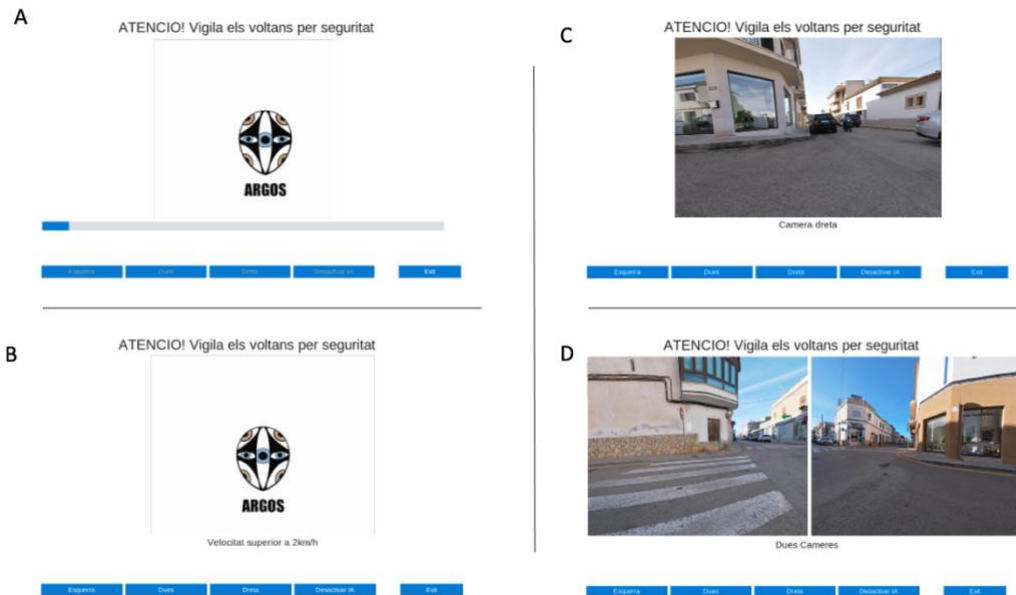
Once the object recognition algorithm used has been defined, it will be explained how the data obtained in it is now processed. The objects that are recognized, i.e., the permitted classes, are: vehicles (bicycles, buses, cars, motorcycles) and people. If an object is not in the classes mentioned above, it would not be detected. Later in the discussion section, we will discuss what would happen if an object that is not on the recognized list appeared.

When any object of the set objects is recognized, for the first time since the car stops at an intersection, a noise will sound to warn the driver. This will not sound again until some class is no longer detected while the vehicle is still stopped or once the car moves again and stops at a new intersection.

In addition to the audible warning, the device has light warnings to identify if what has been detected is a pedestrian or a vehicle and which side it is on. This will remain active as long as one of the allowed classes is detected. In the next section you can see in more detail how you will see the output that the driver will receive.

## 6.6 USER INTERFACE AND ALERTS

As mentioned above, PySimpleGUI has been used for the creation of the user interface. The user will be able to interact with the GUI while the car is moving at a speed of less than 2 km/h or is stopped, and will be able to decide which of the cameras is seen. In addition to this, a button has been incorporated that allows you to disable the recognition of objects in case the driver only wants to have the cameras active and not the warnings. Below are the different screens that the user will see:



**Figure 18 GUI screens. A) Charging screen B) Screen while the vehicle is in motion C) Screen with a single camera D) Screen with both cameras**

The luminous and sound warnings are integrated into the electronic circuit shown in Figure 19. This will be activated when any of the relevant objects are recognized. There are four light signals, 2 on each side. In this way, the driver can quickly know on which side, the object has been detected. Light signals are divided into two categories: people and vehicles (cars, motorcycles, bicycles, and trucks).

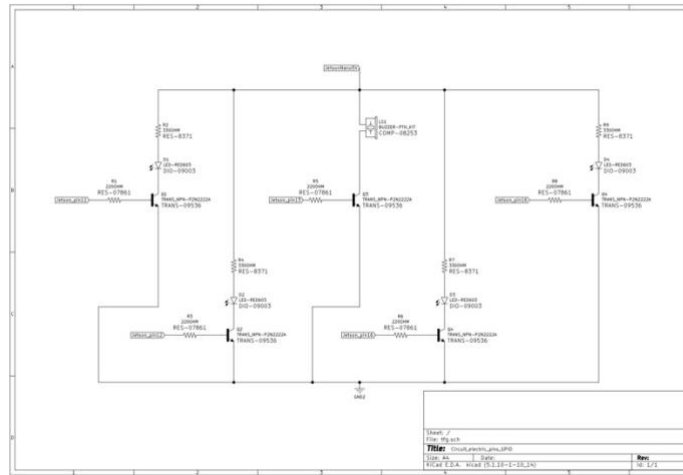


Figure 19: Electronic circuit generated with KiCad

To encapsulate the project, a housing has been designed for the screen, battery, embedded system, and electronic board, and a different one for each camera. This task was carried out using the CAD Fusion360 program. The design blueprints of the different pieces have also been generated.

In images 20 and 21, the different parts of the design are shown. In Annex 4 you can find the design plans for each piece.



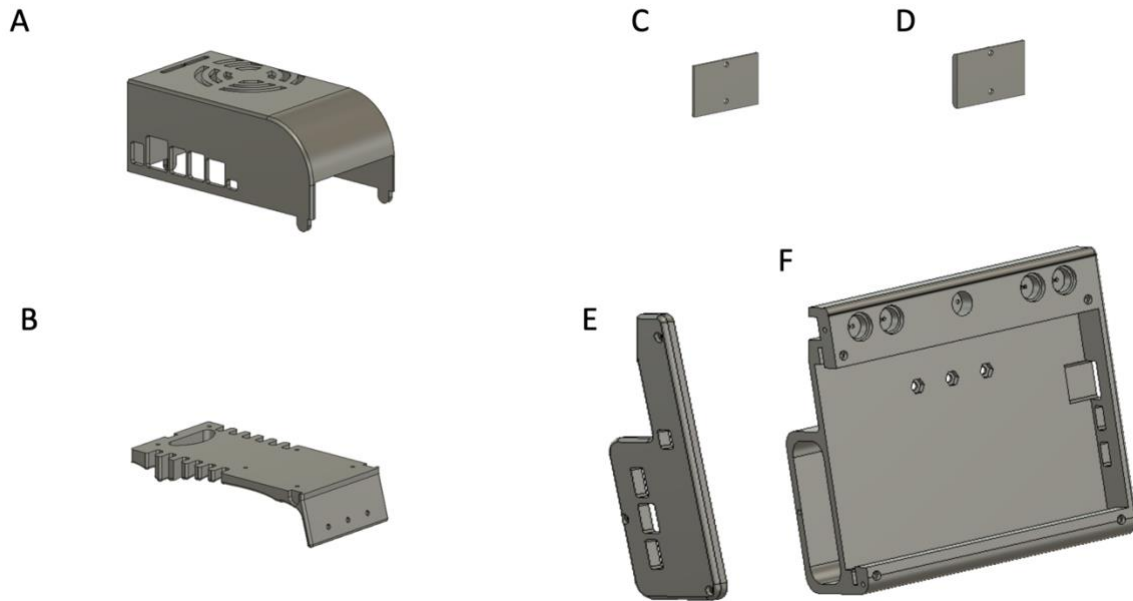


Figure 20: Parts of the screen housing, battery, and embedded system A) Embedded system, housing cover B) Base housing for embedded system, C) and D) Rear cover housing for screen E) Screen and battery housing

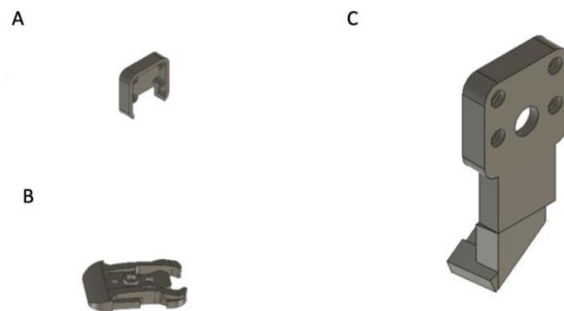


Figure 21: Right camera housing.

A) Rear housing B) Clip to hook up to the car C) Front housing

## 7. RESULTS

In order to evaluate the results involving image processing, 12 videos have been recorded in different street intersections, with the cameras placed in their position, in order to create a data set for the validation of the proposed algorithms. Of these, 5 videos have been randomly selected from which 4000 frames have been collected. Of the 4000 images, 300 of them were randomly selected.

### 7.1 VEHICLE DATA COLLECTION

The collection of the data is carried out quite accurately. This is because the signal sent to the tachometer is the same as that read by the OBD reader. In image 22, you can see what the screen looks like when the car is going at a speed of more than 2 km/h.

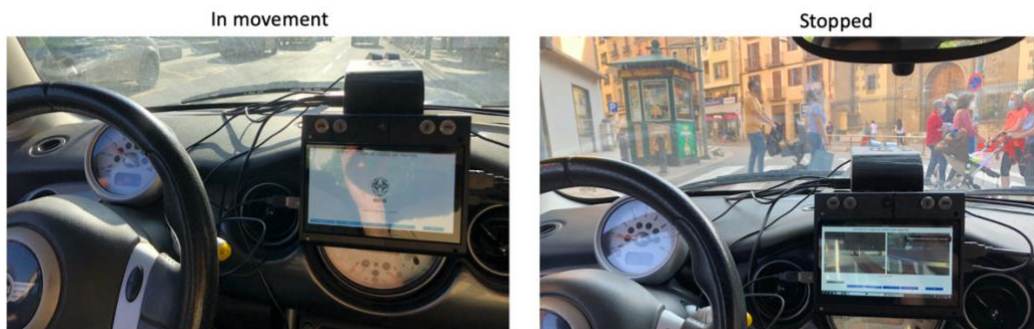


Figure 22: Example of the GUI with the car stopped and with the car in motion

The connection between the board and the car could not be established due to an error with the board acquired. But to test the operation of the reader, a computer has been used with this it has been verified that the operation of this was adequate and the function of collecting the speed of the car and activating and deactivating the screen was fulfilled.

It should be noted that there is a small delay, of 4 seconds, due to serial communication between the computer and the reader. This is not of the utmost importance and does not significantly affect the outcome of the project. The main consequence is the GUI takes this longer to change.

## 7.2 IMPROVED VISIBILITY

Once the different images of the date set and the videos have been reviewed, it can be said that the positioning of the cameras allows to achieve the proposed objective of reducing the distance at which the car invades the street, thus improving the visibility of the driver and reducing the risk of accident.

In the following image, you can see an example from the controller's point of view of what the image would look like on the device.

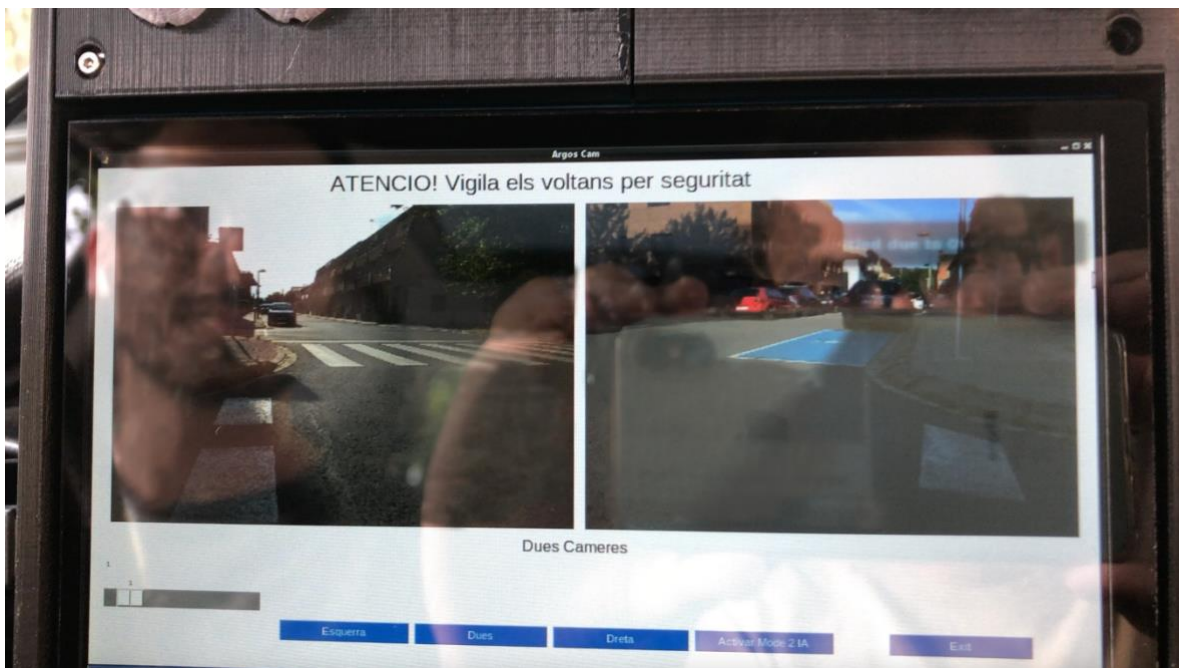


Figure 23: View of the cameras from the driver's point of view

## 7.3 BACKGROUND SUBTRACTION

This section will present the results of the selection of the **background subtraction** algorithm and some examples of the images obtained after it *has been applied*.

To select and evaluate the **background subtraction** algorithm, the set of images mentioned at the beginning of the section has been used. In order to compare the results obtained with those expected, the ground *truth* has been obtained using Photoshop, as indicated in the methodology section. The following illustration presents an example of the masks obtained compared to ground *truth*.

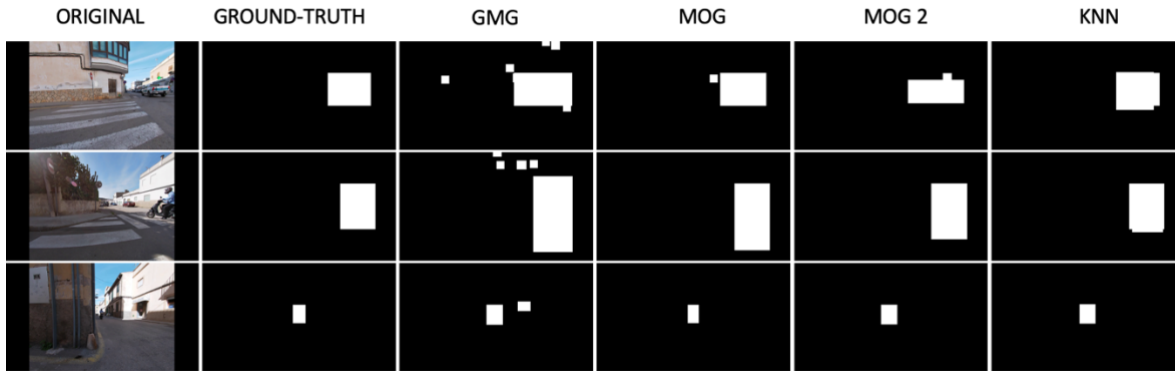


Figure 24 subtraction algorithms

### 7.3.1 SELECT ALGORITHM

Table 1 presents the results obtained from the evaluation of the different algorithms of *Background Subtraction*. For the evaluation of each of the algorithms, the masks - segmented manually and described above - have been used as ground *truth*. The table presents the measurements used for the evaluation that have been presented to the methodology: **Accuracy** (correctly classified pixel ratio), **Sensitivity** (True positives with respect to all positives), **Specificity** (Number of samples classified as negative with respect to all samples), **F-1** (sensitivity-to-accuracy ratio) and processing time (time it takes to process each *frame*)

Table 1 Background subtraction algorithms

BS algorithm	Accuracy	Sensitivity	Specificity	F-1 value	Processing time(s)
KNN	0,983	0,883	0,922	0,853	0,0015
Mog	0,913	0,599	0,736	0,543	0,003
MOG2	0,973	0,785	0,711	0,648	0,002
GMG	0,950	0,375	0,931	0,474	0,0025

Table 1 shows that all algorithms have a high precision value, but as mentioned earlier, this value can be affected by unflagged data. With this in mind, it is necessary to evaluate the other measures.

Being a project for which the most important thing is that the positives are detected correctly, that is, objects that are in motion, sensitivity is a parameter that must be considered. This need is due to the fact that a minor error could end up causing an accident, for example, in the event that an approaching car from the side was not detected and - as a result of not receiving any notification - a collision could occur. In this sense, the cost of making an FN vs FP type error is much higher.

According to this, the MOG and GMG algorithms can be discarded, as they do not detect these objects correctly. The GMG algorithm has a very high specificity but in the application use of this project is not so imported. In fact, if you look at the F-value it is the lowest of all algorithms.

Finally, between the MOG2 and the KNN, it has been determined that KNN is the algorithm that works best of all those that have been tested, taking into account that the value of F-1 is greater than the rest in addition to having the highest specificity. The values obtained in this evaluation coincide with those obtained by other studies carried out [38],[13].

---

### 7.3.2 K-NN BACKGROUND SUBTRACTION AND POSTPROCESSING

The evaluation of this algorithm has taken place in the selection process described in the previous section. This is because, in all previous cases, the images used for evaluation had gone through all the processing steps since the image was obtained in the *bounding box*.

In any case, a brief qualitative assessment of the process is presented below. The following images present different situations to which the **background subtraction** algorithm has been applied. In some of the images presented you can see how static elements are isolated from the images, such as parked cars and other elements that do not need to be recognized. In other cases, you see how elements that are in motion are detected, but should not be detected, such as reflections and shadows.

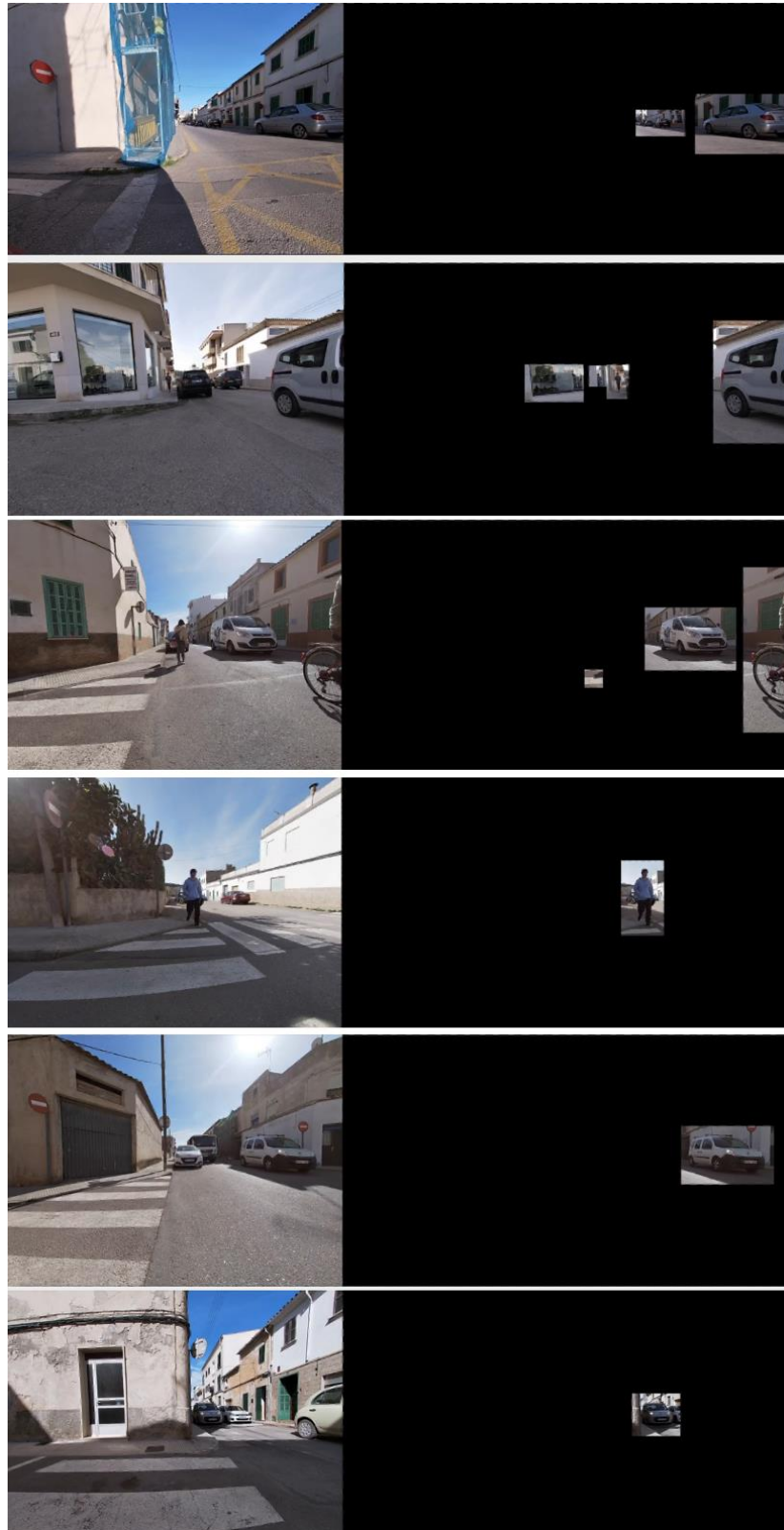


Figure 25: Examples of applied background subtraction



## 7.4 OBJECT DETECTION

### 7.4.1 SELECT ALGORITHM

This section explains why this method of object detection was selected, as well as the advantages it has over others. Before the decision to use the Mobilenet -SSD v2 network, tests have been carried out with the object detector YOLO(You only look once)[22]: these two networks are the most used by embedded systems. As mentioned earlier in the vehicle data collection, to detect vehicles moving at a speed of 60 km/h, imaging must take place at a speed of more than 17 FPS. With this in mind, the YOLOv3 algorithm should be discarded, as it would only allow recognizing objects moving at 18 k/h. In Table 2, the execution speeds (in FPS) of the algorithms and mAP of each algorithm are compared. <sup>4</sup>[39]

**Table 2** Comparison of object detection algorithms [35],[11],[22], [40]

Algorithm	Execution Speed (FPS)	mAP	Dataset
YOLOv3	5	0.7423	VOC2007 [41]
		0,606	MS-COCO [34]
Tiny YOLO	25	0,331	MS-COCO
MobileNet-SSD v2	39	0,727	VOC0712
		0,68	MS-COCO

Another very important factor is the correct detection of objects, in this case vehicles and people. Misdetecion of these could cause an accident. Taking into account the data in the table, and comparing the two remaining networks, it is determined that the best algorithm for this use case is the MobileNet-SSD v2 network, since it presents a better accuracy than tinyYOLO.

### 7.4.2 OBJECT DETECTION WITH MOBILENET-SSD V2

<sup>4</sup> **mAP:** It is about the *mean average precision(mAP)*, a widely used measure for evaluation algorithms of recognition of objects. *The average precision - (AP)* is calculated by searching the area under the curve of **accuracy** and **recall**. The mAP is the average AP of all the classes.

The obtained results for the detection of objects have been affected by different elements.

First of all, the execution speed (in FPS) obtained with the Jetson Nano is reduced compared to those obtained in the computer, due to the more limited processing capacity. Using a GUI increases this decrease in execution speed from 15 FPS to 10 FPS. This happens as a result of the GUI waiting for an event to enter, causing a small delay and therefore a reduction in speed. In Table 3, you can see the different execution speeds in the different situations. Another factor that causes speed reduction is the use of a portable battery or power supply. In this way, it makes the algorithm no longer functional, due to the high loss of frames. It has such an effect that even the detection of people is impossible. But as mentioned in the methodology section, only this feeding method has been used, due to the limitation of the vehicle used for testing. If the device is incorporated into a vehicle it would be done with the power of 10W.

**Table 3** Object recognition processing speed in different situations

Devices					
	Macbook pro, 16GB Ram and i5 processor	Jetson Nano (10W power supply) with GUI, plugged in	Jetson Nano (10W power supply) without GUI, plugged in	Jetson Nano (5W power supply) connected plugged in	Jetson Nano (5W power supply) connected to power bank
Fps	20	10	15	6	2

The evaluation of the object detection was carried out with the set of 300 images of the data set. These have been visually inspected and it has been established whether there were any of the objects to be detected. Each class has been labeled as a binary problem. For the evaluation criterion, three cases have been considered, which will allow the calculation of the accuracy and the **recall**. The first case would be that the result of the algorithm and the visually valued coincided, this would be TP. In the event that the result of the algorithm indicates that the detected object belongs to the wrong class it will be FP and finally if the object is present in the image but has not been detected it will be labeled as FN. It is worth mentioning that there are two classes that could not be evaluated due to lack of data. Table 4 shows the number of samples for each class within the data set of 300 images, and the number of PV, FP and FN for each of them. The table shows that most of the images belong to the person and car classes.



Table 4 Sample number by class and TP, FP, and FN values

	Number of samples with present object	TP	FP	FN
car	128	78	6	50
motorcycle	24	8	0	16
person	122	80	10	42
bicycle	0	0	0	0
bus	0	0	6	0

With the values of PV, FP and FN, the following table has been created where you can see the values of **precision** and recall obtained by each class.

Table 5 Results of object recognition evaluation

	Precision	Recall
car	0.92	0.68
motorcycle	1	0.33
person	0.93	0.65
bicycle		Not enough images
bus		Not enough images

In the table above you can see that the recognition is carried out with great precision. This indicates that, of all the detections that have been made, a large portion has been successful. On the other hand, the *recall* indicates that of all the detections that the model should have had, there have been a large number that have not been detected. After analyzing the results and reviewing the images, it has been possible to detect that the main problem in cases in which the object has not been detected, is due to the distance it is with respect to the camera. The algorithm starts having problems with detection when the object is about 7 meters from the camera, this can be a problem and would require a deeper evaluation to see if it means that the system is not effective.

## 7.5 USER INTERFACE AND ALERTS

Once the data is analyzed, a series of images are then displayed where you can see how the light warnings are activated when a recognized object is detected. Remember that it is only activated on the side where the object has been detected. The first example shows how, when a person is detected, the LED is activated with the person symbol. In the following image you can see the two active LEDs simultaneously. And finally in the last image you can see how an LED is active on either side of the screen.



Figure 26: Examples of light warnings in different situations

Finally, image 27 shows the device mounted on the car. This has been attached to the dashboard of the car with Velcro.

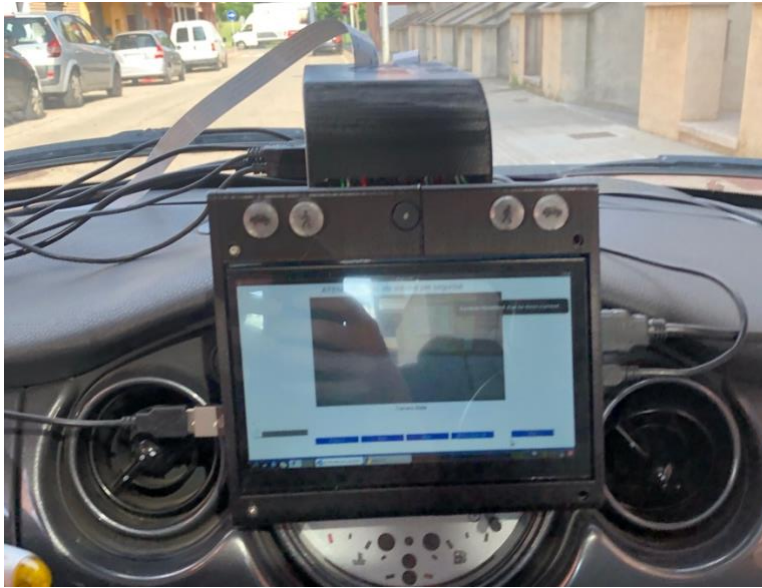


Figure 27 ARGOS cam system mounted on a car

## 8. DISCUSSION

The difference in distance that the car invades the street means greater safety because it is not invading the street that the driver wants to join, which reduces the risk of collision. The quality of the images displayed by the screen, although not of high quality, is sufficient so that the objects that were intended can be recognized: vehicles and people. However, in adverse conditions, such as heavy rain, fog or a poorly lit area at night or if the camera is in backlight, it can mean that the captured images are rendered useless, and the object is not detected correctly. These adversities could be overcome, although a more comprehensive study would already be needed to find a suitable method.

In relation to the *Background Subtraction* module, the resulting algorithm has presented results that fulfill their function of isolating static objects such as parked cars. Masking static subjects can also lead to problems in some cases where a person remains static. In image 24, you can see an example. If this occurs, it will also be masked and will not be detected by the object detection algorithm. In other cases, reflections are also detected, as these translate into movement in the image. These do not produce a negative effect; they only increase the detected areas.

To solve the problem, of the people who remain static, it would be necessary to try to modify the algorithm, for example, combining it with another object detector to recognize only

people or make use of a *Semantic Segmentation* algorithm such as those presented in the state of the art.

As for image recognition, it has been shown that it is possible to achieve detection with adequate FPS and with high precision allowing to detect objects correctly, as long as it works with a power of 10W and is powered at 5V 4A. The main problem is that this detection cannot be carried out with the selected board if the graphical interface is used, since it reduces the fps by half, with an output value lower than desired. In addition, the low recall results obtained, and the FN cases observed during labeling are an indication of how much the neural network needs to be improved so that it can recognize objects at a greater distance.

Another point to note is that, if an object that is not in the classes of objects mentioned would not be recognized, and this could cause a dangerous situation. An example could be: if a ball was placed in the middle of the street, it would not be detected, but at the first moment that a person appeared behind it the algorithm would detect it. Another example could be an animal that was placed in the middle of the street, unless it is followed by a recognized object would not be detected and the driver would not be notified, which could end up causing an accident. Given these examples, it may be appropriate to expand the number of detected classes by introducing common objects that may be in the middle of the road.

## 9. CONCLUSION

In conclusion, with the realization of this project, it has been shown that it is possible to manufacture a device, which makes use of image processing, which allows to improve visibility in the corners and in this way make possible the reduction of this type of accidents.

This device could be incorporated into systems that are already integrated into most cars today, such as screens, cameras and other sensors that would facilitate its construction. In the ideal case this could become a new ADAS that incorporates the cars.

The most problematic part of the whole project has been the correct configuration of the Jetson Nano, since I wasn't familiarized with the Linux operating system and the installation of packages on it, in addition to the incompatibilities that have been encountered with the libraries with which I had previously worked. Another point that has caused problems, has been the drastic reduction of FPS observed when passing the code to the board, due to the limited multiprocessing capacity that it has presented and the limitation of the processing power, due to the power supply of the power bank.

The three objectives proposed at the beginning of the project have been largely met. The only thing that has not been able to be fully complied with, has been the complete integration of the system in the car, but knowing the origin of the problem, it is simple to solve, powering the board in a more efficient way.

In short, for the system to be incorporated into a car as part of a new ADAS system, it would have to go through a series of improvements. Below are possible future improvements that could be applied to the project so that it can be a new driver assistance system.

### 9.1 FUTURE MILLORES

#### 9.1.1 VEHICLE DATA COLLECTION

In the case of integration into a vehicle, the speed reading would be done internally, making it more accurate.

#### 9.1.2 IMAGE CAPTURE



For example, for the first two you could try to use the sensor fusion technology, used by some autonomous cars, which has been discussed in the state-of-the-art section. The solution would be to combine the reading of cameras and distance sensors .

The second problem could be to try to solve by changing the type of cameras used by night vision cameras. This would already mean the addition of infrared sensors to allow visibility.

---

### 9.1.3 EMBEDDED SYSTEM

The use of a higher version of the Jetson Nano, such as the Jetson Nano Xavier NX, allows for higher processing speed, improving system performance in all scenarios.

The most important improvement would be to connect the board to a source that had the right voltage and currents. Direct connection to the car battery with a 12 to 5 V step-down converter would be one of the best options.

---

### 9.1.4 OBJECT DETECTION

To improve this section, *transfer learning* techniques could be used to retrain the neural network used, applied to the situation presented by the project, in which the camera is in the corner. In addition, you could also train with images in adverse situations such as those presented in the results section, to see if you get an improvement without having to change the hardware.

## 10. BIBLIOGRAPHY



- [1]"The Grimoire of Beasts: Argos."  
<https://grimoriodebestias.blogspot.com/2014/04/argos.html> (accessed May 11, 2021).
- [2] DGT, "Tables Statistics 2019", 2019. Retrieved: 15 May 2021. [Online].  
<https://www.dgt.es/es/seguridad-vial/estadisticas-e-indicadores/accidentes-30dias/tablas-estadisticas/2019/>.
- [3]"ADAS: Past, Present and Future | Vehicle Service Pros."  
<https://www.vehicleservicepros.com/service-repair/diagnostics-and-drivability/article/21198482/adas-past-present-and-future> (accessed May 11, 2021).
- [4]"What are car surround vision cameras and why are they better than they should be? - ExtremeTech." <https://www.extremetech.com/extreme/186160-what-are-surround-view-cameras-and-why-are-they-better-than-they-need-to-be> (accessed May 11, 2021).
- [5]"Lidar vs Camera — Which is best for autonomous cars? | by Vincent Tabora | 0xMachina | Medium." <https://medium.com/0xmachina/lidar-vs-camera-which-is-the-best-for-self-driving-cars-9335b684f8d> (accessed May 11, 2021).
- [6]"Auto Workshop – History of obd." <https://ibtaller.com/historia-del-obd/> (accessed May 11, 2021).
- [7]"OBD system of a car: everything you need to know - motor channel." <https://www.motor.mapfre.es/consejos-practicos/consejos-de-mantenimiento/como-funciona-el-sistema-obd/> (accessed May 11, 2021).
- [8]"Comparing the Different Interface Standards for Embedded Vision," 14/11/18.  
<https://www.automate.org/blogs/comparing-the-different-interface-standards-for-embedded-vision> (accessed May 12, 2021).
- [9]"Beginner's Guide: Choose the Right Camera Modules for Your Raspberry Pi or Jetson Nano Development Kit." <https://www.arducam.com/choose-camera-modules-raspberry-pi-jetson-nano-guide/> (accessed May 12, 2021).
- [10] D. Reifs, "Embedded Systems".
- [11]"Jetson Nano: Deep Learning Inference Benchmarks | NVIDIA Developer." <https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks> (accessed April 21, 2021).
- [12]"Buy a Raspberry Pi 4 Model B – Raspberry Pi." <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/> (accessed May 27, 2021).
- [13] T. Trnovský, P. Sýkora and R. Hudec, "Comparison of Background Subtraction Methods on Near Infra-Red Spectrum Video Sequences," in *Procedia Engineering*, January 2017, vol. 192, pp. 887–892, doi: 10.1016/j.proeng.2017.06.15.
- [14] D. Zeng, X. Chen, M. Zhu, M. Goesele and A. Kuijper, "Background Subtraction with Real-time Semantic Segmentation."
- [15]"Semantic Segmentation - MATLAB & Simulink."





<https://www.mathworks.com/solutions/image-video-processing/semantic-segmentation.html> (accessed May 13, 2021).

- [16] "What is LiDAR technology?" <https://blog.generationrobots.com/en/what-is-lidar-technology/> (accessed February 14, 2021).
- [17] "Deep Learning: what is it go to be a technology clave in the future of artificial intelligence." <https://www.xataka.com/robotica-e-ia/deep-learning-que-es-y-por-que-va-a-ser-una-tecnologia-clave-en-el-futuro-de-la-inteligencia-artificial> (accessed May 13, 2021).
- [18] K. L. Masita, A. N. Hasan and T. Shongwe, "Deep learning in object detection: A review," August 2020, doi: 10.1109/icABCD49160.2020.9183866.
- [19] D. Sarkar and Towards Data Science, "A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning | by Dipanjan (DJ) Sarkar | Towards Data Science," 14/11/18. <https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a> (accessed May 13, 2021).
- [20] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5)". Retrieved: 13 May 2021. [Online]. Available: <http://www.cs.berkeley.edu/~rbg/rcnn>.
- [21] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." Retrieved: 13 May 2021. [Online]. <http://image-net.org/challenges/LSVRC/2015/results>.
- [22] "YOLO: Real-Time Object Detection." <https://pjreddie.com/darknet/yolo/> (accessed April 21, 2021).
- [23] W. Liu *et al.*, "SSD: Single Shot MultiBox Detector." Retrieved: 21 April 2021. [Online]. <https://github.com/weiliu89/caffe/tree/ssd>.
- [24] "SSD Object Detection: Single-Shot Multibox Detector for Real-Time Processing | by Jonathan Hui | Medium." <https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06> (accessed May 13, 2021).
- [25] "Elm327-Interface Usb Cable V1.5 For Car Diagnostic,Compatible With All Obd2 Protocols For Windows Elm 327,Scanner Obd Usb - Buy Elm327 Usb V1.5,Usb Cable Interface,Obd Scanner Product on Alibaba.com." <https://spanish.alibaba.com/product-detail/elm327-usb-v1-5-car-diagnostic-usb-cable-interface-supports-all-obd2-protocols-for-windows-elm-327-usb-obd-scanner-697370759.html?spm=a2700.galleryofferlist.0.0.73c03ca2WshEaa> (accessed May 27, 2021).
- [26] "Getting Started - python-OBD." <https://python-obd.readthedocs.io/en/latest/> (accessed May 14, 2021).
- [27] "Raspberry Pi Camera Module V2.1 | Raspberry Pi camera module, CSI-2 interface, resolution 3280 x 2464 pixels, 30fps | RS Components." <https://es.rs-online.com/web/p/camaras-para-raspberry-pi/9132664/> (accessed May 17, 2021).





- [28]"opencv\_contrib/evaluation.py master · opencv/opencv\_contrib · GitHub."  
[https://github.com/opencv/opencv\\_contrib/blob/master/modules/bgsegm/sample\\_s/evaluation.py](https://github.com/opencv/opencv_contrib/blob/master/modules/bgsegm/sample_s/evaluation.py) (accessed April 18, 2021).
- [29]"Background Subtraction with OpenCV and BGS Libraries | Learn OpenCV."  
<https://learnopencv.com/background-subtraction-with-opencv-and-bgs-libraries/>  
(accessed April 18, 2021).
- [30] Z. Zivkovic and F. Van Der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction," *Pattern Recognit. Lett.*, vol. 27, no. 7, pp. 773–780, May 2006, doi: 10.1016/j.patrec.2005.11.005.
- [31]"OpenCV: cv::BackgroundSubtractorKNN Class Reference."  
[https://docs.opencv.org/4.5.2/db/d88/classcv\\_1\\_1BackgroundSubtractorKNN.html](https://docs.opencv.org/4.5.2/db/d88/classcv_1_1BackgroundSubtractorKNN.html)  
(accessed April 18, 2021).
- [32]"KNN Algorithm - Finding Nearest Neighbors - Tutorialspoint."  
[https://www.tutorialspoint.com/machine\\_learning\\_with\\_python/machine\\_learning\\_with\\_python\\_knn\\_algorithm\\_finding\\_nearest\\_neighbors.htm](https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_knn_algorithm_finding_nearest_neighbors.htm) (accessed May 14, 2021).
- [33]"OpenCV: Morphological Transformations."  
[https://docs.opencv.org/4.5.2/d9/d61/tutorial\\_py\\_morphological\\_ops.html](https://docs.opencv.org/4.5.2/d9/d61/tutorial_py_morphological_ops.html)  
(accessed April 18, 2021).
- [34]"COCO - Common Objects in Context." <https://cocodataset.org/#home> (accessed April 21, 2021).
- [35]"GitHub - chuanqi305/MobileNet-SSD: Caffe implementation of Google MobileNet SSD detection network, with pretrained weights on VOC0712 and mAP=0.727."  
<https://github.com/chuanqi305/MobileNet-SSD> (accessed April 21, 2021).
- [36] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications."
- [37]"MobileNet version 2." <https://machinethink.net/blog/mobilenet-v2/> (accessed April 21, 2021).
- [38]"[OpenCV Current Combat] 43 Use OpenCV for background segmentation - Programmer Sought." <https://www.programmersought.com/article/10826881459/>  
(accessed April 18, 2021).
- [39]"mAP (medium accuracy) could confuse you! | by Shivy Yohanandan | Towards Data Science." <https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2> (accessed May 31, 2021).
- [40]"GitHub - taehoonlee/tensornets: High-level network definitions with weights previously trained in TensorFlow." <https://github.com/taehoonlee/tensornets> (accessed April 21, 2021).
- [41]"The PASCAL Visual Object Classes Homepage."  
<http://host.robots.ox.ac.uk/pascal/VOC/> (accessed April 21, 2021).
- [42] M. Grundmann, V. Kwatra, and I. Essa, "Self-directed video stabilization with robust L1 optimal camera paths," *Proc. IEEE Comput. I'm Conf. Comput. Vis. Recognition*





## 11. ANNEX

Annex 1: More examples of corners where the project can be applied.....	52
Annex 2:Image processing class.....	61
Annex 3: Main code .....	68
Annex 4:Plans for the 3D design of the project.....	69

## 11.1 CORNERS



Annex 1: More examples of corners where the project can be applied

## 11.2 CODE MADE WITH PYTHON

As mentioned in the methodology, the code consists of two parts. The first is a class created to control the camera algorithms and the second is responsible for managing the GUI, reading the OBD and integrating the cameras into the system.

### 11.2.1 IMAGE PROCESSING CLASS

```
#The camera threading script has some parts based on this code
# MIT License
# Copyright (c) 2019,2020 JetsonHacks
# See license
# A very simple code snippet
# Use of two CSI cameras (such as the Raspberry Pi Version 2) connected to a
# NVIDIA Jetson Nano Developer Kit (Rev B01) using OpenCV
# Drivers for camera and OpenCV are included in the base image in JetPack 4.3+

# This script will open a window and place the camera sequence from each camera in a window
# arranged horizontally.
# The camera sequences are each read on its own thread, as when sequentially there is
# is a noticeable delay
# For better performance, the next step would be to experiment with having the window screen
# on a separate thread

#Import libraries
thread importing threading
importing threads
import cv2 as cv, cv2
amount of time
numpy as np amount
bone import
amount RPi.GPIO as GPIO
jetson.inference amount as inference
amount jetson.utils as utilities

#create a class to defines the camera
vStream class:
    #class inicialitization
    def __Init__(self, net, Width height, board ,pin_led_person, pin_led_car, pin_buzzer):

#camera parameters (source, width i height)

    #GPIO #
    #GPIO pins parameters definition
```



```
self.board = dish
self.pin_led_person = pin_led_person
self.pin_led_car = pin_led_car
self.pin_buzzer=pin_buzzer
#decalration of GPIO pins, and set them to LOW
self.board.setup(self.pin_led_car, self.plaque.OUT, initial=self.board.LOW)
self.board.setup(self.pin_led_person, self.plaque.OUT, initial=self.board.LOW)
self.board.setup(self.pin_buzzer, self.plaque.OUT, initial=self.board.LOW)
self. threshold_of_frames = 5 frames #numero before changing GPIO state

#Camera parameters #
#Definition of size of the frame
self.width = width
self.height = height
self-portrait = None
self.frame = None
self.grabbed = False
self.running = True

#Neural network #
propio.net = network
self.thr = 0.5 # threshold, 0.5 default
# definition of the Names of the classes that will be recognized
self.classes = ['background', 'truck', 'bicycle', 'bus', 'car', 'motorbike', 'person']
self.classNames = ['person']
self.classVehicle = ['bicycle', 'truck', 'bus', 'car', 'motorbike']
#Parametres to control the status of the GPIO during recognition
self.led_person_On = False
self.led_vehicle_On = False
self.detetcted_person = False
self.detetcted_vehicle = False
self.counter_detected_person = 0
self.counter_not_detected_person = 0
self.counter_detected_vehicle = 0
self.counter_not_detected_vehicle = 0

#Background subtraction #
self.box_increase = 15 #increment of obtained bounding box

Self.color = (255, 255, 255)
#background subtraction and parameter adjustment
self.fgbg = cv2.createBackgroundSubtractorKNN()
self.fgbg.setHistory(100)
self.fgbg.setNSamples(10)
self.fgbg.setkNNSamples(3)

#definition of morphological transofrmations
self.kernel = cv2.getStructuringElement(cv2. MORPH_CROSS, (5, 5))
self.kernel1 = cv2.getStructuringElement(cv2. MORPH_ELLIPSE, (7, 7))
```



```
#Thread #
self.read_thread = None
self.read_lock = Threading.Lock()
self.running = False
```

```
#Function to initialize the camera
def open(self, gstreamer_pipeline_string):
    try:
        # the video driver will be gstramer
        self.destratus = cv2.VideoCapture(gstreamer_pipeline_string, cv2.CAP_GSTREAMER)
        self.grabbed, self.frame = self.capture.read()

    except RuntimeError: #warns if there is an error opening the camera
        self.portrait = None
        print("Impossible to open camera")
        print("Pipeline: " + gstreamer_pipeline_string)
        return
```

#Funcio to start the thread, call the function to start reading frames

```
def start(self):
    if self.running:
        print("It's already capturing Video")
        return None

    if self.running != None:
        self.running = True
        self.read_thread = Threading.Thread(target=self.updateCamera)
        self.read_thread.daemon = True
        self.read_thread.start()

    return
```

#function to stop the thread, necessary to do cleaning when the program is stopped

```
def stop(self):
    self.running = False
    self.read_thread.join()
```

#function to update the camera and get new frames

```
def updateCamera(self):
    while self.running:
        try:
            grabbed, frame = self.capture.read()
            with self.read_lock:
                self.grabbed = grabbed
                self.frame = frame
            try:
                self.frame2 = cv2.resize(self.frame, (300, 300)) # Change frame size
```





```

except Exception as error: #check for errors
    print("error in camera capture:", err)

except RuntimeError:
    print("Could not read the camera")

# function to set all the GPIO to low and only read the frame
def getFrame(self):
    print("frame")
    self.board.output(self.pin_led_car, self.board.LOW)
    self.board.output(self.pin_led_person, self.board.LOW)
    self.board.output(self.pin_buzzer, self.board.LOW)
    return self.frame2

#Function to apply background subtraction
def Background_subtraction(self):
    try:
        self.blur= cv2. GaussianBlur(self.frame2, (15, 15), 0) # blur the image
        self.fgmask = self.fgbg.apply(self.blur) # Process background mask
        self.binary = cv2.threshold(self.fgmask, 50, 255, cv2. THRESH_BINARY) # Ensures that masks it is
        binary
        self.open = cv2.morphologyEx(self.binary, cv2. MORPH_OPEN, self.kernel) # Reduces image
        noise
        self.close= cv2.morphologyEx(self.open, cv2. MORPH_CLOSE, self.kernel1) # makes the mask
        look cleaner
        self. dilate = cv2.morphologyEx(self.close, cv2. MORPH_DILATE, self.kernel1) # Expands mask

        #first find contours, if necessary a second one is applied
        self.contours, self.hierarchy = cv.findContours(selfexle. dilate, cv.RETR_TREE,
        cv.CHAIN_APPROX_SIMPLE)
        self.contours_poly = [None] * Len(self.contours)
        self.boundRect = [None] * Len(self.contours)
        for self.i, .c yourself in enumerate(self.contours):
            self.contours_poly[self.i] = cv.approxPoliDP(.c self, 3, True)
            self.boundRect[self.i] = cv.boundingRect(self.contours_poly[self.i])

        self.drawing = np.zeros((self.frame2.shape[0], self.frame2.shape[1], 3), dtype=np.uint8)

        for self.i in range(Len(self.contours)):
            cv.rectangle(self.drawing, (int(self.boundRect[self.i][0]) - self.box_increase),
                (int(self.boundRect[self.i][1])) - self.box_increase),
                ((int(self.boundRect[self.i][0] + self.boundRect[self.i][2]) + self.box_increase),
                (int(self.boundRect[self.i][1] + self.boundRect[self.i][3])) + self.box_increase),
                self.color, cv2. FILLED)

        self.draw = cv2.cvtColor(self.drawing, cv2. COLOR_BGR2GRAY)

        # second find contours, applied in case 2 bounding boxes are in contact and merges them
        self.contours_, self.hierarchy_ = cv.findContours(self.draw, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
        self.contours_poly = [None] * Len(self.contours_)

```





```

self.boundRect = [None] * Len(self.contours_)
self.size = 20 #filtra the Oer Outlines the size
self.pass_filter = [] # llist of contour with area greater than filter size
for self.j, self.co in enumerate(self.contours_):
    self area = cv2.contourArea(self.co)
    if self area > self.size:
        self.pass_filter.append(self.j)
        self.contours_poly[self.j] = cv.approxPoliDP(self.co, 3, True)
        self.boundRect[self.j] = cv.boundingRect(self.contours_poly[self.j])

self.drawing_ = np.zeros((self.frame2.shape[0], self.frame2.shape[1], 3), dtype=np.uint8)

#Filter if contours were found
if len(self.contours_) > 0:
    for self.h on self.pass_filter:
        cv.drawContours(self.drawing_, self.contours_poly, self.h, selfcolor)
        cv.rectangle(self.drawing_, ((int)self.boundRect[self.h][0]) - self.box_increase),
(int(self.boundRect[self.h][1])) - self.box_increase), \
        ((int)self.boundRect[self.h][0] + self.boundRect[self.h][2]) + self.box_increase),
        (int(self.boundRect[self.h][1] + self.boundRect[self.h][3])) + self.box_increase),
self.color, cv2. FILLED)

self.res_box = cv2.bitwise_and(self.frame2, self.frame2,
                                mask=cv2.cvtColor(self.drawing_, cv2. COLOR_BGR2GRAY))
more:
self.res_box = cv2.bitwise_and(self.frame2, self.frame2,
                                mask=cv2.cvtColor(self.drawing, cv2. COLOR_BGR2GRAY))

self.res_box return

except exception exception failure: #check for errors
    print("Background subtraction error", failure)

#function to apply object recognition with background subtraction
def recognition_with_bg(self):
    try:
        #applies BG
        self.res_box=self. Background_subtraction()
        self.background_resized= cv2. resize(self.res_box, (300, 300)) #resize the frame
        self.frame_copy = self.frame2.copy()
        self.background_resized = cv2.cvtColor(self.background_resized, cv2. COLOR_BGR2RGBA)

        # transformin image into Cuda format
        self.background_resized = utils.cudaFromNumpy(self.background_resized)
        self.cols = self.background_resized.shape[1]
        self.rows = self.background_resized.shape[0]
        self.detections = self.net.Detect(self.background_resized) #optimized object detection with
cuda

self.detetcted_person = False

```



```
self.detected_vehicle = False
for self.detection in self.detections: #loop that goes through all detections obtained
    self.ID = self.detect.ClassID
    self.item = self.net.GetClassDesc(self.ID)

    self.confidence=self.detect.Confidence

    if self.confidence > self.thr and (self.item in self.classes): #filter by class and threshold
print(self.item)

# GPIO Control
# If one of the Leds is not active and the counter detected is less than 5 it's going to
produce a sound warning and start the light
if self.item in self.classVehicle:
    self.detected_vehicle=True
    if (not self.led_vehicle_On) and self.counter_not_detected_vehicle >
self.threshold_of_frames: # if the LED is not ON
    #the led will be turn on
    print("buzzer")
    self.board.output(self.pin_buzzer, self.board.HIGH)
    self.led_vehicle_On = True
    self.counter_not_detected_vehicle = 0
    self.board.output(self.pin_led_car, self.board.HIGH)
    print("vehicle led turned on")

if self.item in self.classNames:
    self.detected_person = True
    if (not self.led_person_On) and self.counter_not_detected_person >
self.threshold_of_frames: # if the LED is not on

    self.led_person_On = True
    self.counter_not_detected_person = 0
    self.board.output(self.pin_led_person, self.board.HIGH)
    print("Person led turned on")

#self.board.output(self.pin_buzzer, self.board.LOW)

if self.detected_person: #increase detected number
    self.counter_detected_person += 1
    if self.counter_detected_person > 2:
        self.board.output(self.pin_buzzer, self.board.LOW)
        print("Detected", self.counter_detected_person)

else: ##increase not detected number
    self.counter_not_detected_person += 1
    print("not Detected", self.counter_not_detected_person)
    if self.counter_not_detected_person>2:
        self.board.output(self.pin_buzzer, self.board.LOW)
```



```
if self.detected_vehicle: #increase detected number
    self.counter_detected_vehicle += 1
    if self.counter_detected_vehicle > 2:
        self.board.output(self.pin_buzzer, self.board.LOW)
        print("Detected", self.counter_detected_vehicle)

else: #increase not detected number
    self.counter_not_detected += 1
    print("not Detected", self.counter_not_detected_vehicle)
    if self.counter_not_detected > 5:
        self.board.output(self.pin_buzzer, self.board.LOW)

#if not self.detected: # if the LED is on and not detected goes over the threshold
if (not self.detected_vehicle) and self.counter_detected_vehicle > self.threshold_of_frames:
    if self.led_vehicle_On:
        print("led vehicle off")
        self.board.output(self.pin_led_car, self.board.LOW)
        self.led_vehicle_On = False
        self.counter_detected_vehicle = 0

if (not self.detected_person) and self.counter_detected_person > self.threshold_of_frames:
    if self.led_person_On:
        print("person led off")
        self.board.output(self.pin_led_person, self.board.LOW)
        self.led_person_On = False
        self.counter_detected_person = 0

return self.frame_copy, self.frame2, #returns original frame
except Exception as error_rec: error #error check
    print("Error" recognition", error_rec)

#object recognition without background subtraction
def recognition_without_bg(self):
    try:
        # instead take the background output Take the camera frame
        self.background_resized = cv2.resize(self.frame2, (300, 300))
        self.frame_copy = self.frame2.copy()
        self.background_resized = cv2.cvtColor(self.background_resized, cv2.COLOR_BGR2RGBA)
        self.background_resized = utils.cudaFromNumpy(self.background_resized)
        self.cols = self.background_resized.shape[1]
        self.rows = self.background_resized.shape[0]

        #apply object recognition with cuda
        self.detections = self.net.Detect(self.background_resized)

board.output(self.pin_buzzer, self.board.LOW)

#if not self.detected: # if the LED is on and not detected goes over the threshold
```



```
if (not self.detected_vehicle) and self.counter_detected_vehicle > self.threshold_of_frames:
```

```
    if self.led_vehicle_On:
```

```
        print("led vehicle off")
```

```
        self.board.output(self.pin_led_car, self.board.LOW)
```

```
        self.led_vehicle_On = False
```

```
        self.counter_detected_vehicle = 0
```

```
if (not self.detected_person) and self.counter_detected_person > self.threshold_of_frames:
```

```
    if self.led_person_On:
```

```
        print("person led off")
```

```
        self.board.output(self.pin_led_person, self.board.LOW)
```

```
        self.led_person_On = False
```

```
        self.counter_detected_person = 0
```

```
    return self.frame_copy, self.frame2, #returns original frame
```

```
except Exception as error_rec: error #error check
```

```
    print("Error" recognition", error_rec)
```

```
#funcio to define the Raspberry Camera Parameters on the Jetson Nano
```

```
Def gstreamer_pipeline(
```

```
    sensor_id=0,
```

```
    sensor_mode=3,
```

```
    capture_width=1280,
```

```
    capture_height=720,
```

```
    display_width=640,
```

```
    display_height=480,
```

```
    framerate=30,
```

```
    flip_method=0,
```

```
):
```

```
    return (
```

```
        "nvarguscamerasrc sensor-id=%d sensor-mode=%d ! "
```

```
        "video/x-raw(memory:NVMM), "
```

```
        "width=(int)%d, height=(int)%d,"
```

```
        "format=(string)NV12, framerate=(fraction)%d/1 ! "
```

```
        "nvvidconv flip-method=%d ! "
```

```
        "video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! "
```

```
        "videoconvert ! "
```

```
        "video/x-raw, format=(string)BGR ! appsink"
```

```
    % (
```

```
        sensor_id,
```

```
        sensor_mode,
```

```
        capture_width,
```

```
        capture_height,
```

```
        frame rate,
```

```
        flip_method,
```

```
        display_width,
```

```
        display_height,
```

```
    )
```

```
)
```

Annex 2 Image processing class

---

### 11.2.2 MAIN CODE

```
#imported libraries
import PySimpleGUI as sg
from camaras_sincronizadas_reconocimiento_modos import vStream, gstreamer_pipeline
import numpy as np
import cv2 as cv, cv2
import time
from obd.protocols import ECU
from obd import OBDStatus
import obd
from queue import Queue
import threading
import Jetson.GPIO as GPIO
import jetson.inference as inference
import jetson.utils as utils
```

```
obd.commands.TIME_SINCE_DTC_CLEARED.ecu = ECU.ALL  
que = Queue()
```

```
# definition of speed to be used as a global  
Speed = 0  
high_speed=True  
exit=True
```

```
#function to create connection with the OBD reader
```

```
def connect(n):
```

```
    n = "Good"
```

```
    fast_ = True
```

```
    ports = obd.scan_serial() # search USB ports, and return the available ones
```

```
    print(ports) #shows the available ports
```

```
    #establish the asynchronous connection to be able to read the values of the car
```

```
    connection = obd.Async(ports[0],baudrate=38400, protocol="3", fast=fast_, timeout=30)
```

```
    while Len(connection.supported_commands) < 10: #try to connect until more than 10 commands  
        are supported
```

```
try:
```

```
    connection.stop()
```

```
    time.sleep(5)# stops the thread 5 seconds to try to connect
```

```
    connection = obd.Async(ports[0],baudrate=38400, protocol="3", fast=fast_, timeout=30)
```

```
except Exception as error:
```

```
    print(error)
```

```
print("ends")
```

```
if OBDStatus.CAR_CONNECTED: #check if connection was successful
```

```
    Print("related")
```

```
connection return
```

```
#function to get speed from serial
```

```
def new_value1(speed):
```

```
    #declaration of global variables
```

```
    global high_speed
```

```
    global outputs
```

```
    global speed
```

```
    speed = speed.value.magnitude # assign value to speed
```

```
    #condicional to regulate the GUI
```

```
    if speed < 2:
```

```
        high_speed = False
```

```
    else:
```

```
        high_speed = True
```



```
# function that acts as a counter for the progress bar
def time(second, window window, thread):
    progress = 0
    for i in range(Int(second * 10)):
        time.sleep(1) # sleep for a while
        progress += 100 / (second * 10)
        window['-SEC-'].click()

    if not (thread.is_alive()): #check if other thread has finished
        window['-THREAD-'].click()
        break
    Print("threading1", progress)

window['-THREAD-'].click()
Print("finished_2") #check thread stop

def main():
    sg.theme('Reddit')

    # ----- Create the Designs that make up the screens-----
    GIF = "parche.png" #imatge of the loading screen
    Height = 400
    wide_1 = 650
    wide_2 = 490
    #Layout screen loading
    layout1 = [[Sg. image(Name=GIF, size=(height, height), background_color='White', key='-IMAGE-')],
                [Sg. Progress Bar(30, orientation='h', size=(100, 20); key='progbar')],
                ]

    # design for the Left Camera
    design2 = [
        [Sg. image(Name=GIF, key='image1', pad=(200, 0), size=(wide_1, height))],
        [Sg. Text message('left camera', size=(40, 1), justification='center', font='Helvetica 20')]]

    # design for two Cameras
    layout3 = [[Sg. image(Name=GIF, key='image2', size=(wide_2, height)),
                Sg. image(Name=GIF, key='image3', size=(wide_2, height))],
                [Sg. Text message('Two cameras', size=(40, 1), justification='center', font='Helvetica 20')]]

    # design for the right camera
    layout4 = [
        [Sg. image(Name=GIF, key='image4', pad=(200, 0), size=(wide_1, height))],
        [Sg. Text message('right camera', size=(40, 1), justification='center', font='Helvetica 20')]]

    layout_movment = [
        [Sg. image(Name=GIF, background_color='White', key='-IMAGE2-', pad=(200, 0), size=(wide_1,
height))],
        [Sg. Text message('Speed greater than 2 km/h', size=(40, 1), justification='centre',
source='Helvetica 20')]]
```

```
#buttons layout
layout5 = [[Sg. button('Left', size=(20, 1), pad=(2, 2), font='Helvetica 14', visible=true, Disabled=true),
# change by icon
Sg. button('Two', size=(20, 1), pad=(2, 2), font='Helvetica 14', visible=true, Disabled=true),
# change by icon
Sg. button('Ok', size=(20, 1), pad=(2, 2), font='Helvetica 14', visible=True, disabled=True),
Sg. button(button_text='disable AI', size=(20, 1), pad=(2, 2), font='Helvetica 14', visible=True,
disabled=True, key='-IA-'),
Sg. button('Exit', size=(20, 1), pad=(50, 2), source='Helvetica 14', visible=True)]]

# main design, combines the above layouts
design = [[Sg. Text message('ATTENTION! Check the surroundings for safety, justification='c',
source='Helvetica 30',
pad=(125, 0), size=(40, 1), key="top text")],
[Sg. column(layout1, justification="t", element_justification='center', key='-COL1-'),
Sg. column(layout2, justification="t", element_justification='center', visible=False, key='-COL2-
'),
Sg. column(layout3, justification="t", element_justification='center', visible=False, key='-COL3-
'),
Sg. column(layout4, justification="t", element_justification='center', visible=False, key='-COL4-
'),
Sg. column(movement_layout, justification="t", element_justification='center', visible=False,
key='-COL5-')],
[Sg. button("-SEC-", visible=False), Sg. button('-THREAD-', visible=False)],
[Sg. column(layout5, justification="t", element_justification='center', visible=True, key='-COL6-
')]]

#defines the window with correct dimensions
window = Sg. window('Argos Cam', layout, resizable=False, size=(1024, 600)). Finish()

timeout = thread = none

speed_up_progress_bar = False
timeout = None
t = 50 # maximum startup time in seconds
c = "a"

#threads for Loading Screen
thread = Threading. thread(target=lambda q, arg1: q.put(connect(arg1)), Args=(that, c),
daemon=True)
thread1 = Threading. thread(goal=Time, Args=(t, window, thread), daemon=True)
thread.start()
thread1.start()

#element to continue loading bar if thread finishes
time_ = 0
#loop the Loads screen
while True: # Event Loop
    event, values = window.read(timeout = 10)
```





```
if event on (None, 'Exit', 'Cancel') or time_ > 100: #if times up or exit button, loop ends
break
```

```
if event == '-SEC-': internal #click invisible button to update progress bar
    #if speed_up_prosgress_bar is true the program loads faster
    if speed_up_progress_bar:
        time_ = time_ + 1
        window['progbar'].update_bar(time_, 100)
        time.sleep(.1)
        window['-SEC-'].click()
    more:
        time_ = time_ + 1
        window['progbar'].update_bar(time_, 100)
```

```
# finishes the thread and assign connection variable
if event == '-THREAD-':
    #stop threads
    thread.join(timeout = 0)
    connection = que.get()
    #connection.watch(obd.commands.SPEED, callback=new_value1)
    #connection.start()
    thread1.join(timeout=0)
    print('Thread Finished', thread.is_alive(), thread1.is_alive())
    speed_up_progress_bar = #aumenta true speed of bar loads
    window['-SEC-'].click()
    # rest
```

```
#defineix camera dimensions in windows according to the selected mode
dispW = 640
dispH = 480
altura_ = 400
wide_1_ = 650
wide_2_ = 490
```

```
#defineix used GPIO Pins
Jetson = GPIO
Jetson.setmode(GPIO.BCM)
pins = [18, 17, 7, 27, 10]
# seven pins as output pin with optional HIGH status
Jetson.setup(pins, GPIO.OUT, initial=GPIO.LOW)
```

```
#inicialitza the web of nerves that enters chambers
network = inference.detectNet("ssd-mobilenet-v2", threshold=0.5)
```

```
#es create the Vstream Objects with the Required Parameters i Start capturing Frames
camera1 = vStream(network, dispW, dispH, Jetson, 18, 17, 7)
camera1.open(gstreamer_pipeline(flip_method=2))
camera1.start()
camera2 = vStream(network, dispW, dispH, Jetson, 27, 10, 7)
camara2.abringstreamer_pipeline(sensor_id=1, flip_method=2))
camera2.start()
```

```
# Starts by capturing the speed value
connection.watch(obd.commands.SPEED, callback=new_value1)
connection.start()

# Starts capturing the speed value
window['f'-COL1-'].update(visible=False)
layout_visible = 3
window['f'-COL{layout_visible}-'].update(visible=True)

# Makes visible the buttons, to change the camera
window['Left'].update(disabled=False)
window['Dos'].update(disabled=False)
window['Success'].update(disabled=False)
window['-IA-'].update(disabled=False)

# camera size
dim_1 = (wide_1_, altura_)
dim_2 = (wide_2_, altura_)
old_slider = 10
movement_layout = True

#variable to change active algorithm
smart = 2 #0 off, 1 without BG, 2 with Bg

# Main loop
while True:
    event, values = window.read(timeout = 20)
    #event values = window.read()
    if event == 'Exit' or event == Sg. WIN_CLOSED or cv2.waitKey(1) == Ord(
        'q'): # if window closed or exit button clicked the program ends
        break

####change active camera when button click
    elif event == 'Left':
        # disable current layout and activate the proper layout
        window['f'-COL{layout_visible}-'].update(visible=False)
        layout_visible = 2
        window['f'-COL{layout_visible}-'].update(visible=True)

    elif event == 'Two':

        window['f'-COL{layout_visible}-'].update(visible=False)
        layout_visible = 3
        window['f'-COL{layout_visible}-'].update(visible=True)

    elif event == 'right':
        window['f'-COL{layout_visible}-'].update(visible=False)
        layout_visible = 4
```



```
window[f'-COL{layout_visible}-'].update(visible=True)
```

```
#####
```

```
elif event == '-AI-':  
    # selection of algorithm, and button text change  
    if smart == 0:  
        window['-IA-'].update('enable AI Mode 2')  
        smart = 1
```

```
elif smart == 1:  
    window['-IA-'].update('disable IA')  
    smart = 2
```

```
else:  
    window['-IA-'].update('enable AI Mode 1')  
    smart = 0
```

```
if high_speed:  
    # Modifies layout according to the speed of the vehicle  
    window[f'-COL{layout_visible}-'].update(visible=False)  
    window['-COL5-'].update(visible=True)  
    movement_layout = True
```

```
else:  
    window['-COL5-'].update(visible=False)  
    window[f'-COL{layout_visible}-'].update(visible=True)  
    movement_layout = False
```

if movement\_layout == False: # if car is moving at a speed below 2km/h the device will capture images

###read frames with the appropriate algorithm and layout

```
if layout_visible == 2: # shows the left camera  
    if smart == 2:  
        recognition, frame = camara1.recognition_with_bg()
```

```
elif smart == 1:  
    recognition, framework = camara1.recognition_without_bg()
```

```
else:  
    frame = camera1.getFrame()
```

```
# encoding image to use it in pysimplegui  
imgbytes = cv2.imencode('.png', cv2.resize(frame, dim_1))[1].tobytes()  
window['image1'].update(data=imgbytes)
```



```
elif layout_visible == 4: # shows the right camera
    if smart == 2:
        attribution1, frame1 = camara2.recognition_with_bg()

    elif smart == 1:
        attribution1, frame1 = camara2.recognition_without_bg()

    else:
        frame1 = camera2.getFrame()

    imgbytes1 = cv2.imencode('.png', cv2.resize(frame1, dim_1))[1].tobytes() # ditto
    window['image4'].update(data=imgbytes1)

else:# shows both Cameras
    if smart == 2:
        recognition, frame = camara1.recognition_with_bg()
        attribution1, frame1 = camara2.recognition_with_bg()

    elif smart == 1:
        recognition, framework = camara1.recognition_without_bg()
        attribution1, frame1 = camara2.recognition_without_bg()

    else:
        frame = camera1.getFrame()
        frame1 = camera2.getFrame()

    imgbytes = cv2.imencode('.png', cv2.resize(frame, dim_2))[1].tobytes()
    imgbytes1 = cv2.imencode('.png', cv2.resize(frame1, dim_2))[1].tobytes()
    window['image2'].update(data=imgbytes)
    window['image3'].update(data=imgbytes1)

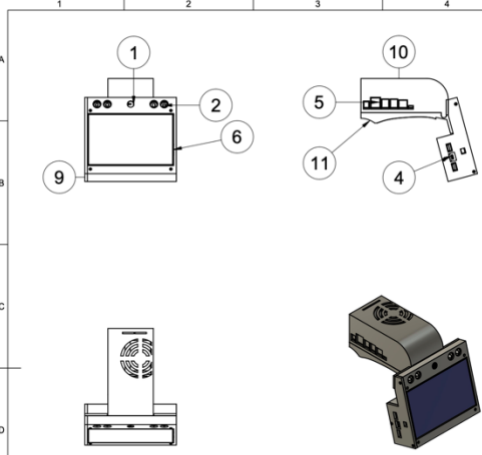
#clean objects use to free memory
window.closing()
camera1.stop()
connection.close()
camera1.capture.release()
camera2.stop()
camera2.capture.release()
GPIO.cleanup()

if __name__ == '__main__':

    main()
```

#### Annex 3: Main code

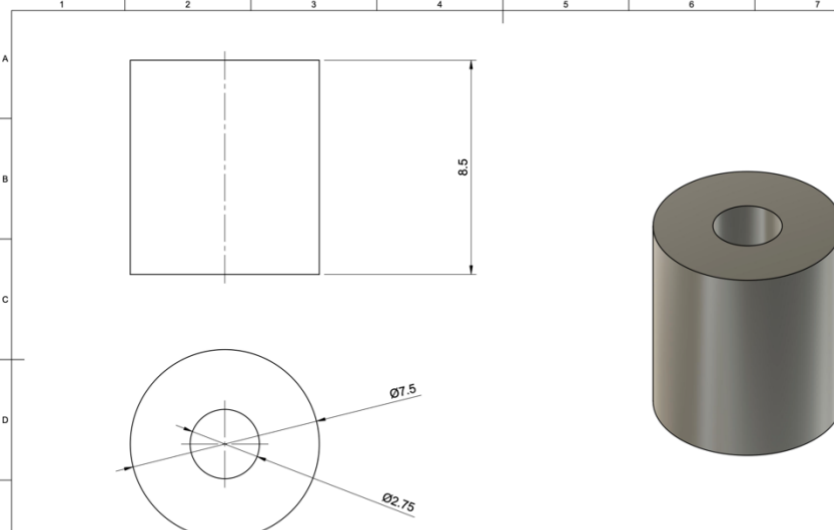
## 11.3 DESIGN PLANS



23	8	Cargol M2x8	
22	3	Cargol M4x10	
21	3	Rosca M4	
20	3	Arandela M4	
19	4	Cargol M2x20	PLA
18	6	Cargol M2x12	PLA
17	10	Rosca M2	PLA
16	10	Arandela Ø2	PLA
15	1	Caixa Pantalla	PLA
14	1	Tapa lateral dreta	PLA
13	1	Tapa lateral esquerra	PLA
12	1	Tapa lateral dreta	PLA
11	1	Base caixa Jetson	PLA
10	1	Tapa caixa jetson	PLA
9	1	Tapa caixa pantalla	PLA
7	1	Spacer(No visible)	PLA
6	1	Pantalla 7" (Internet)	
5	1	Jetson Nano B01 (Internet)	
4	1	Powerbank xiaomi (Internet)	
3	1	Pcb prototipat(Internet)	
2	4	Led(Internet)	
1	1	Buzzer actiu(Internet)	
Numero	Quantitat	Nom part	Material

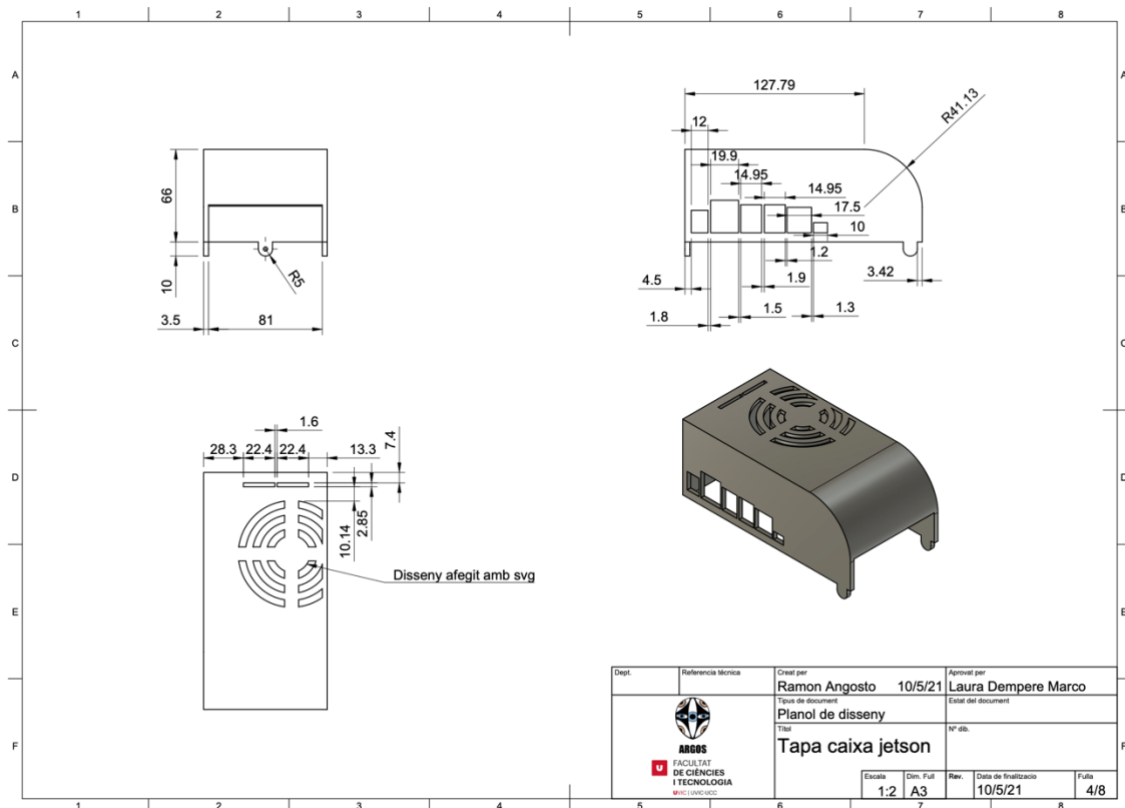
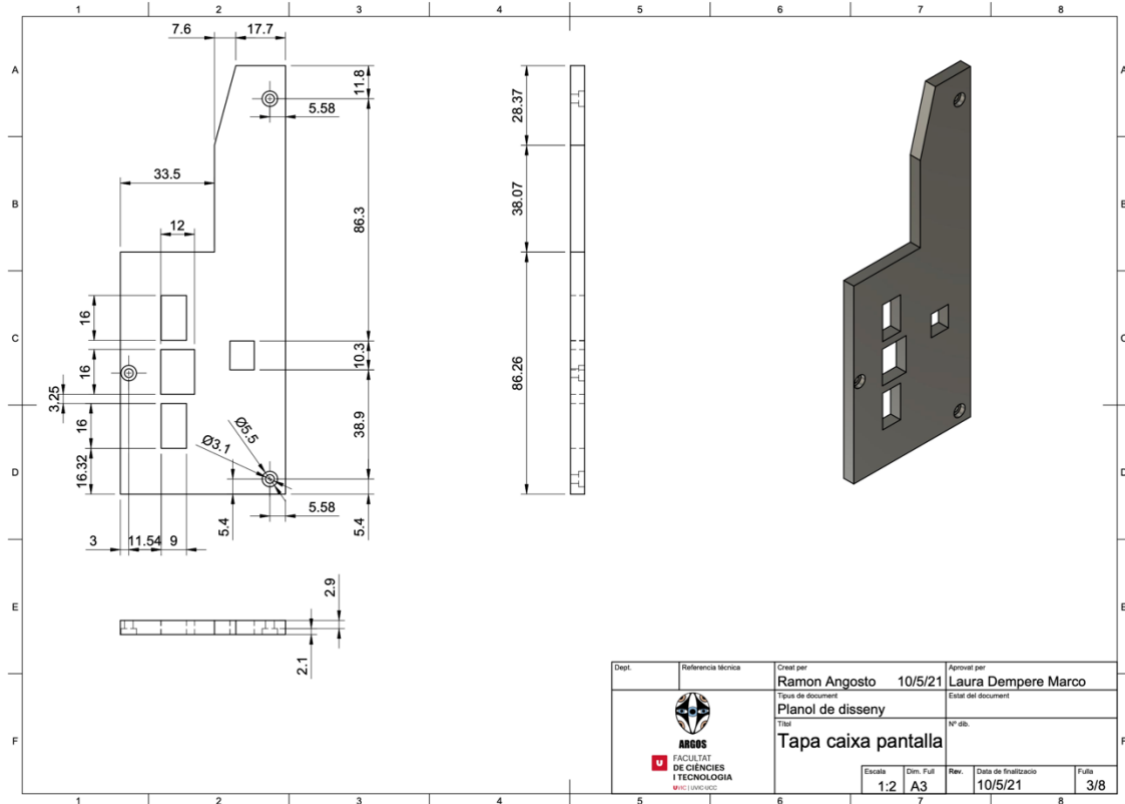
1. Les parts que estan marcades amb (internet) han estat obtingudes de llibreries online, i es referenciaran al treball

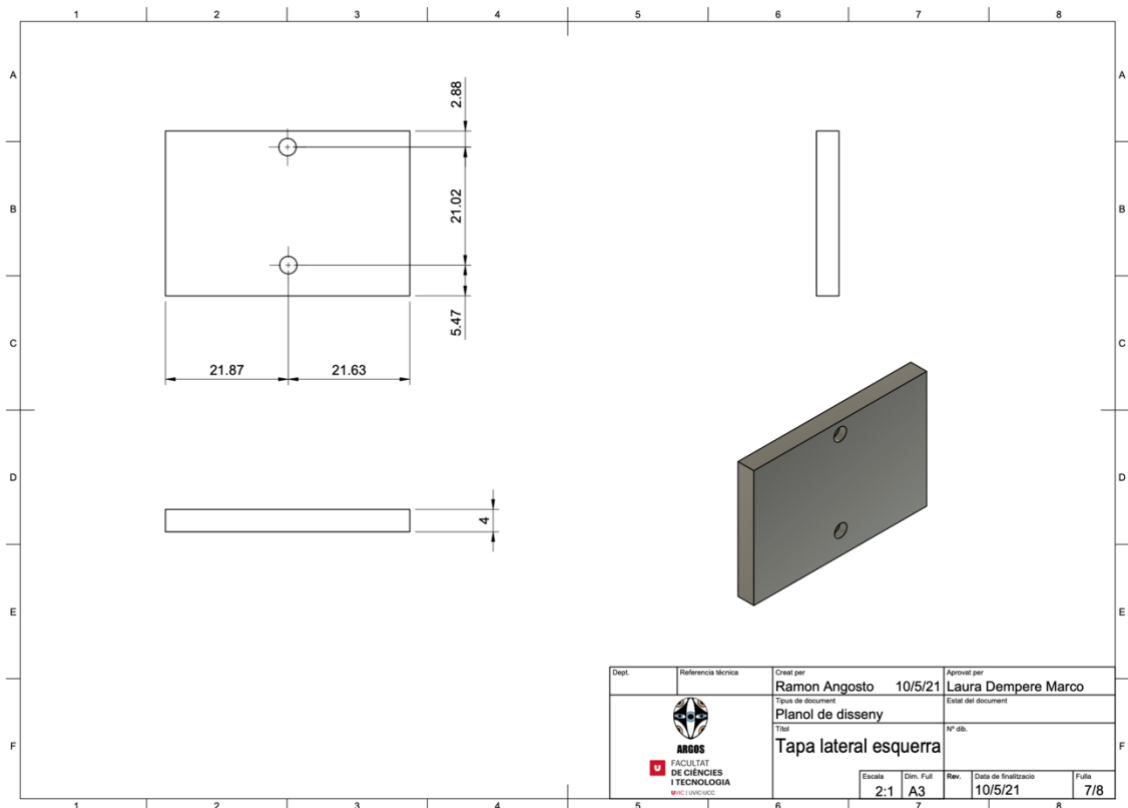
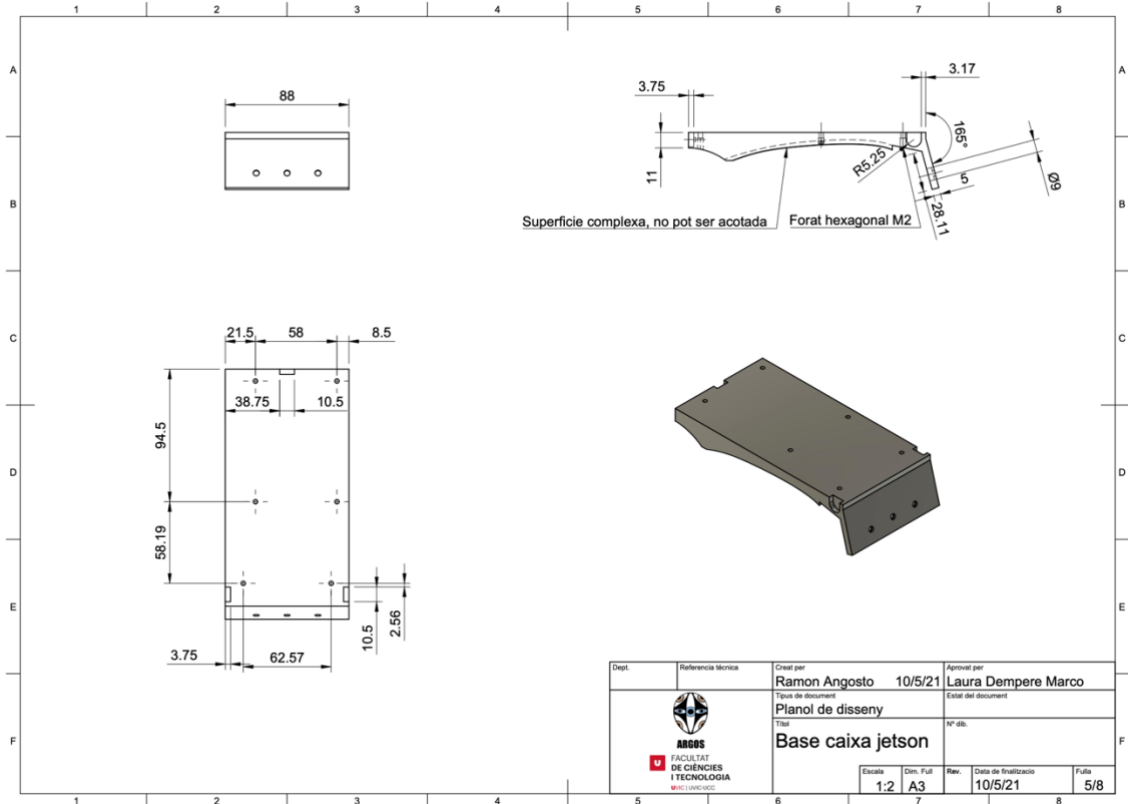
Taula de parts			
Dept.	Referencia tècnica	Disseñat per	Revisat per
		Ramon Angosto	Laura Dempere Marco
Tipus de document		Estat del document	
Ensamblatge		Nº dib.	
Títol		Encapsulat interior cobre	
Escala	Dim. Full	Rev.	Data de finalització
1:5	A3		10/5/21
			Folia
			1/8

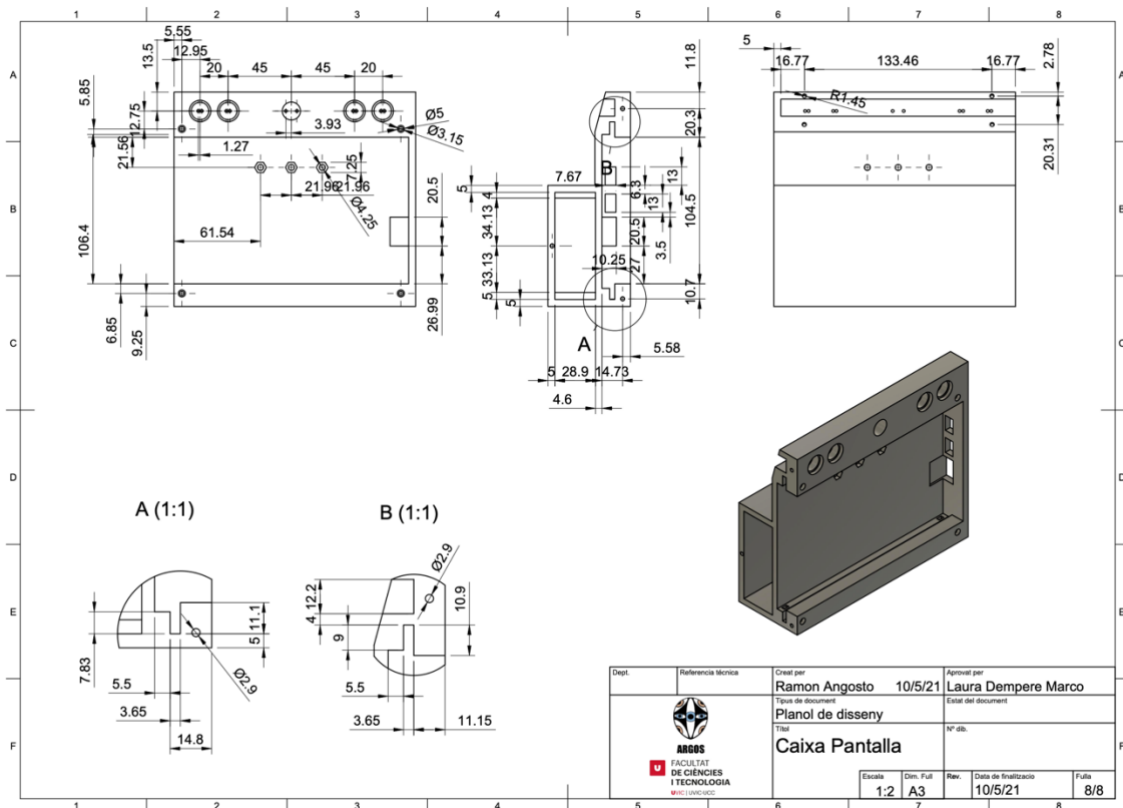
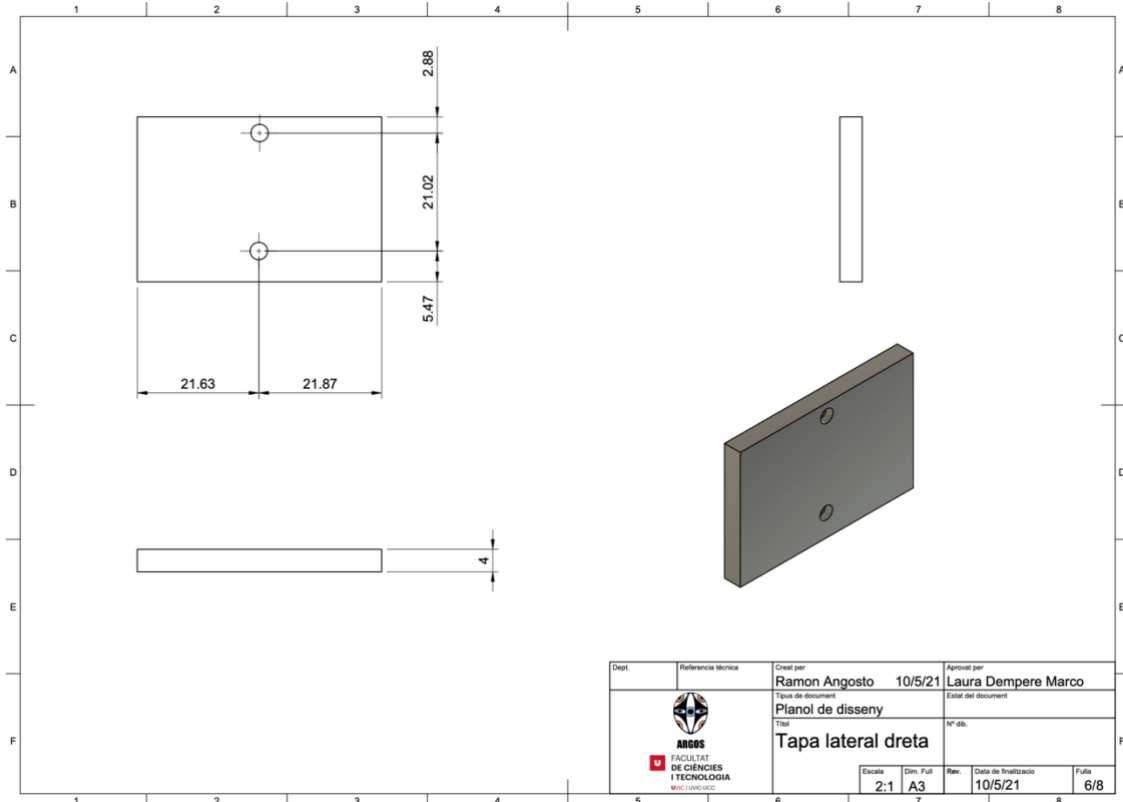


Dept.	Referencia tècnica	Disseñat per	Revisat per
		Ramon Angosto	Laura Dempere Marco
Tipus de document		Estat del document	
Planol de disseny		Nº dib.	
Títol		Spacer	
Escala	Dim. Full	Rev.	Data de finalització
10:1	A3		10/5/21
			Folia
			2/8

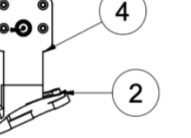
Annex 4Plans for the 3D design of the project

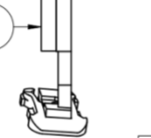





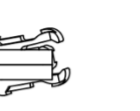













Numero	Quantitat	Nom part	Material
10	2	Cargol M4x10	
9	2	Arandela M4	
8	2	Rosca M4	
7	4	Cargol M2x16	
6	4	Arandela M2	
5	4	Rosca M2	
4	2	Part posterior carcasa	PLA
3	2	Part frontal carcasa	PLA
2	2	Clip Gopro (Internet)	PLA
1	2	Raspi Cam v2 (Internet)	

**Llista de parts**

Dept.	Referència tècnica	Creat per	Aprovat per
	 <b>FACULTAT DE CIÈNCIES I TECNOLOGIA</b> UVIC   UVIC-UCC	Tipus de document <b>Ensamblatge</b> Títol <b>Càmera TFG</b>	Estat del document <b>Acabat</b> N° Dib. 11/5/21
		Escala 1:2 Dim. full A4	Rev. Data acabat 11/5/21

The figure shows a technical drawing of a mechanical part with the following views and dimensions:

- Top View:** Shows a central rectangular body with a width of 21 mm and a height of 18 mm. It has four circular holes arranged in a 2x2 grid. The distance between the centers of the holes is 10.5 mm. The distance from the center of the holes to the top and bottom edges is 5 mm. The distance from the center of the holes to the left and right edges is 12.36 mm. The top and bottom edges are rounded with a radius of R5. The left and right edges are chamfered with a 24° angle. The bottom edge has a 9° angle. The bottom edge has a width of 14 mm.
- Front View:** Shows the part from the front, with a height of 12.36 mm. It has a central rectangular body with a width of 21 mm and a height of 18 mm. It has four circular holes arranged in a 2x2 grid. The distance between the centers of the holes is 10.5 mm. The distance from the center of the holes to the top and bottom edges is 5 mm. The distance from the center of the holes to the left and right edges is 12.36 mm. The top and bottom edges are rounded with a radius of R5. The left and right edges are chamfered with a 24° angle. The bottom edge has a 9° angle. The bottom edge has a width of 14 mm.
- Side View:** Shows the part from the side, with a height of 12.36 mm. It has a central rectangular body with a width of 21 mm and a height of 18 mm. It has four circular holes arranged in a 2x2 grid. The distance between the centers of the holes is 10.5 mm. The distance from the center of the holes to the top and bottom edges is 5 mm. The distance from the center of the holes to the left and right edges is 12.36 mm. The top and bottom edges are rounded with a radius of R5. The left and right edges are chamfered with a 24° angle. The bottom edge has a 9° angle. The bottom edge has a width of 14 mm.
- Isometric View:** Shows the part in a 3D perspective, highlighting its L-shaped profile and the 85° angle between the two main faces.
- Section A-A (1:1):** A cross-section view showing the internal structure of the part, with a width of 5 mm and a height of 3 mm.
- Detail View:** A close-up of the bottom edge, showing a 9° angle and a width of 24.55 mm. It includes the text "Obertura per col·locar rosca M4" (Opening for M4 screw).

