Universidade de Aveiro Departamento de Engenharia Mecânica
2019

Tiago Miguel Rodrigues de Almeida

Arquitetura Multi-Câmara e Multi-Algoritmo para Perceção Visual a Bordo do ATLASCAR2

Multi-Camera and Multi-Algorithm Architecture for Visual Perception onboard the ATLASCAR2

Universidade de Aveiro Departamento de Engenharia Mecânica
2019

Tiago Miguel Rodrigues de Almeida

Arquitetura Multi-Câmara e Multi-Algoritmo para Perceção Visual a Bordo do ATLASCAR2

Multi-Camera and Multi-Algorithm Architecture for Visual Perception onboard the ATLASCAR2

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob orientação científica de Vítor Manuel Ferreira dos Santos, Professor Associado C/ Agregação do Departamento de Engenharia Mecânica da Universidade de Aveiro.

o júri / the jury

presidente / president	**Prof. Doutor Miguel Armando Riem de Oliveira** Professor Auxiliar da Universidade de Aveiro
vogais / committee	**Doutor Eurico Farinha Pedrosa** Bolseiro do *Instituto de Engenharia Eletrónica e Telemática de Aveiro*
	Prof. Doutor Vítor Manuel Ferreira dos Santos Professor Associado C/ Agregação da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Antes de mais agradeço com a máxima sinceridade possível ao Professor Vítor Santos pela forma como me orientou este semestre. Gostaria de deixar realçado também a forma como impôs a sua dedicação, rigor e persistência neste trabalho, o que nos levou ao sucesso.
Depois quero deixar um agradecimento também a todos os que me acompanharam durante os 5 anos mais gratificantes da minha vida. Em particular, ao Bernardo, ao Tavares e ao Pinho por me terem ajudado a crescer em termos de intelecto e pela amizade.
Agradeço também, a ajuda imprescindível do pessoal do LAR, nomeadamente ao Engenheiro Rui Heitor pela parte operacional, ao Bernardo Lourenço por me fazer elevar sempre um pouco mais a fasquia, ao João Santos por todos os problemas resolvidos em conjunto que nos fizeram muitas dores de cabeça e à Daniela pelo companheirismo durante as viagens no ATLAS e durante todo o procedimento de resolução de problemas que nos foi aparecendo pela frente. Obrigado também a ti Manel, pela boa disposição transmitida.
Quero agradecer à minha família pela forma como alicerçaram a minha estadia na Universidade, durante todos estes 5 anos.
Por fim, agradeço também aos meus amigos pela forma como lidaram comigo extra-curricularmente, pela paciência e apreço que demonstraram.

keywords

visual perception, data combination, computer vision techniques, ROS architecture, road lane lines, multiple cameras, multiple algorithms.

abstract

Road detection is a crucial concern in Autonomous Navigation and Driving Assistance. Despite the multiple existing algorithms to detect the road, the literature does not offer a single effective algorithm for all situations. A global more robust set-up would count on multiple distinct algorithms running in parallel, or even from multiple cameras. Then, all these algorithms' outputs should be merged or combined to produce a more robust and informed detection of the road lane, so that it works in more situations than each algorithm by itself. This dissertation integrated in the ATLASCAR2 project, developed at the University of Aveiro, proposes a ROS-based architecture to manage and combine multiple sources of lane detection algorithms ranging from the algorithms that return the spatial localisation of the road lane lines and those whose results are the navigable zone represented as a polygon. The architecture is fully scalable and has proved to be a valuable tool to test and parametrise individual algorithms. The combination of the algorithms' results used in this work uses a confidence based merging of individual detections.

palavras-chave

percepção visual, combinação de dados, técnicas de visão computacional, arquitetura ROS, linhas da via estrada, múltiplas câmaras, múltiplos algoritmos.

resumo

A deteção de estradas é uma questão crucial na Navegação Autónoma e na Assistência à Condução. Apesar de os múltiplos algoritmos existentes para detetar a estrada, a literatura não oferece um único algoritmo eficaz para todas as situações. Uma configuração global mais robusta incorporaria vários algoritmos distintos e executados em paralelo, ou mesmo baseado em múltiplas câmaras. Então, todos os resultados destes algoritmos devem ser fundidos ou combinados para produzir uma deteção mais robusta e informada da via da estrada, para que funcione em mais situações do que cada algoritmo funcionando individualmente. Esta dissertação integrada no projeto ATLASCAR2, desenvolvido na Universidade de Aveiro, propõe uma arquitetura baseada em ROS para gerir e combinar múltiplas fontes de algoritmos de deteção de vias da estrada, desde algoritmos que devolvem a localização espacial da faixa de rodagem até àqueles cujos resultados são a zona navegável representada como um polígono. A arquitetura é totalmente escalável e provou ser uma ferramenta valiosa para testar e parametrizar algoritmos individuais. A combinação dos resultados dos algoritmos utilizados neste trabalho utiliza uma combinação de deteções individuais baseada na confiança.

Contents

List of Tables

Intentionally blank page.

List of Figures

Chapter 1

Introduction

The fields of Autonomous Driving (AD) and Advanced Driver Assistance Systems (ADAS) have developed a wide range of studies that can bring new possibilities to the drivers. When AD achieves all of the aims that are foreseen, it would provide a decrease in the number of accidents on the roads around the world. Also, it will lead to a reduction in traffic jams since its next step is providing communication between cars, which would help in identifying traffic problems early on. Finally, the system relieves the drivers from the daily commuting task, improving their life.

For this, it is important that the car is equipped with all sensors that are determinant to extract information from the real scenario. One of the functions of an autonomous vehicle is to identify the road boundaries to ensure that it is within the lane in order to minimise chances of collisions with other vehicles. Different approaches have been implemented over the years and they are divided into two types: methods that use classic computer vision techniques and more recent approaches that contemplate techniques of learning, based on AI (Artificial Intelligence) [1].

This dissertation is the development of a road boundary detection method that includes a robust architecture, which enables the use of different road detection algorithms to be suitable for both structured and unstructured roads. The former are roads with no lane lines on it, commonly presented on rural locations and the latter are normal roads that have clear lane marks and road boundaries. This is a problem since it is necessary to perceive two different types of features. In addition to this, three main problems have to be considered: lighting change, shadows and vehicle occlusions [2].

1.1 The ATLAS Project

The study inherent in this dissertation is a contribution to the ATLAS project. This project was created by the Group of Automation and Robotics at the Department of Mechanical Engineering of the University of Aveiro, Portugal [3]. It is related to autonomous driving and it aims to develop a sensory architecture that allows the creation of an autonomous car. Therefore, this project has promoted the improvement of advanced active systems and the development of new methods to deal with real road scenarios.

The ATLAS Project started by developing several prototypes of robots, which competed in autonomous driving competitions at the Portuguese National Robotics Festival. Due to the success and the experience acquired in these competitions, the ATLAS Project took a step forward and started by developing the ATLASCAR1 (Fig. 1.1), a full-sized prototype for research on ADAS.

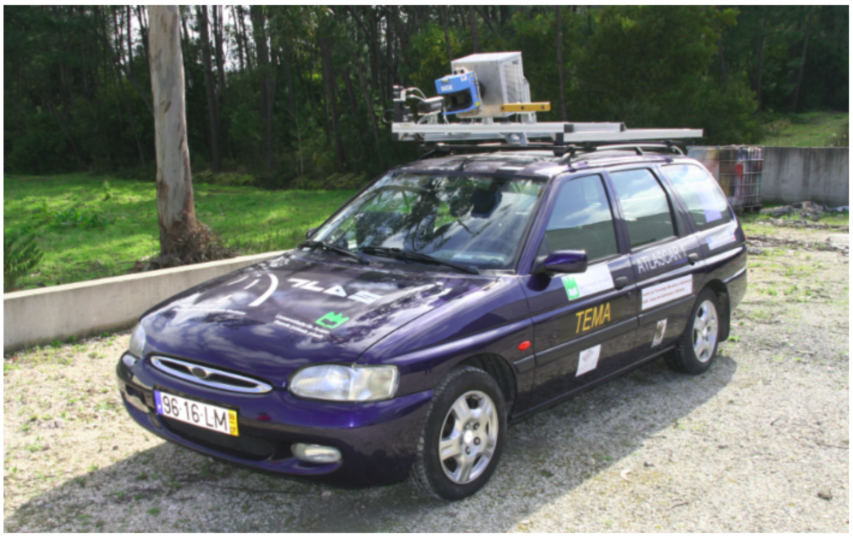


Fig. 1.1: ATLASCAR1 autonomous car [3].

ATLASCAR1 is a Ford Escort Station Wagon from 1998 equipped with a broad range of sensing technologies. These sensors are used to perceive what is happening in its surroundings and identify the current internal status of the vehicle.

After years of studies and implementations of algorithms and hardware, the project replaced the ATLASCAR1 by a new electric vehicle - ATLASCAR2 (Fig. 1.2). It is a Mitsubishi i-MiEV (2015) equipped with 16 kWh batteries and several sensors such as LIDAR sensors, GPS and cameras. This is the vehicle that is going to be used in this dissertation.

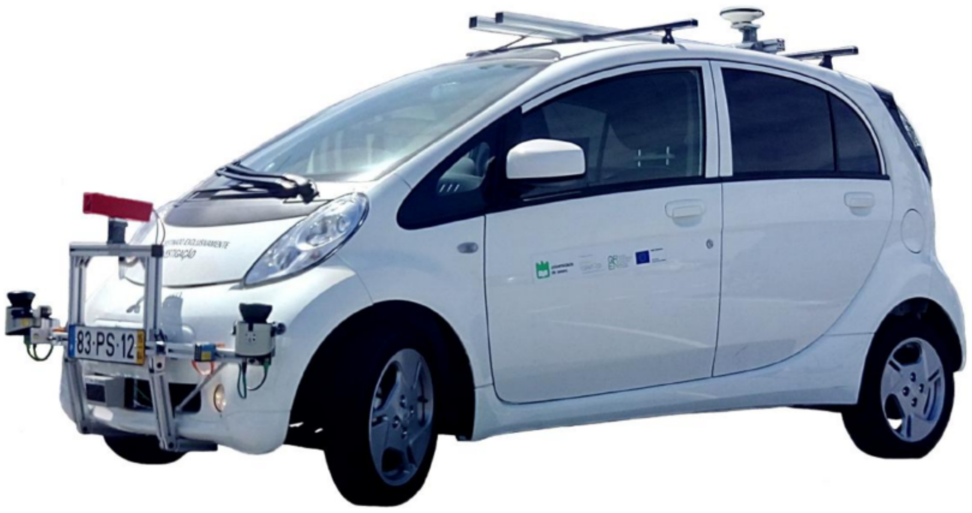


Fig. 1.2: ATLASCAR2 autonomous car.

1.2 Project Context and Motivation

This dissertation is a development of the study of lane markers or road boundaries detectors (when there are no lanes on the road), which has been presented as one of the most complex and crucial features that can be detected by an autonomous car [4]. To achieve a robust boundary detector, the ATLASCAR2 has to be equipped with cameras and it is necessary to conceive a software architecture that is capable of increasing the quality of the planner of the car decision in terms of road boundaries. This idea of detecting lane markers/road boundaries on the ATLASCAR2 is due to if the vehicle is on a flat road, the LIDAR (Light Detection And Ranging) sensors that are implemented are not enough to detect the road boundaries.

Many methods and algorithms can detect road limits, however, there is not a single valid algorithm for all situations and this is why an architecture to combine and fuse them is necessary. The architecture will increase the road boundary detection robustness and, consecutively, the driving safety. This is important because self-driving is a subject that has to be treated carefully since it influences people's security. Therefore, in this dissertation, two Point Grey FL3-GE-28S4-C cameras will be used, which are sensors usually used for this type of study at LAR (Laboratório de Automação e Robótica).

1.3 Objectives

As described in Section 1.2, the installation of cameras and the detection of road boundaries onboard the ATLASCAR2 is a must-have tool on autonomous vehicles and because of that the main objectives of this dissertation are:

- design and install the physical structure that allows cameras to be fixed to the vehicle;
- develop a ROS software architecture based on two crucial premises: scalability and redundancy. The idea is to create a ROS based architecture that can be used when the number of cameras is increased and that allows the use of several algorithms at the same time in order to get more information about the road boundaries features;
- find an effective method of merging the information returned by each algorithm;
- test the system developed in a real case scenario.

1.4 Document Structure

This document is divided into seven chapters. Chapter 1 is a brief explanation of the goals of the dissertation, which were mentioned previously. Chapter 2 provides a summary of the methods implemented by different authors and explains the history of this technology. Chapter 3 describes the hardware and software used in this project; The physical structure designed and developed during the dissertation is also detailed in Chapter 3. Chapter 4 presents the methods used to calibrate the camera. Chapter 5 explains the architecture developed to solve the problem presented. Chapter 6 shows the results obtained after the development of the architecture. Finally, Chapter 7 contains the conclusions about the project and the possible continuity of the work.

Intentionally blank page.

Chapter 2

State of the Art

There are several methods and algorithms developed to detect road lane lines and road boundaries. This section will first explore the history of this kind of technology. After that, works developed at LAR will be presented, followed by methods implemented in other types of contexts.

2.1 Genesis of the Technology

In 1992, Mitsubishi presented a camera-assisted lane-keeping support system on the Mitsubishi Debonair (Fig. 2.1) sold in Japan [5]. The system worked like several current systems: if the driver drifted across those road markings, an alarm would sound to alert the driver. This type of technology is called LDWS (Lane Departure Warning System) and its usage is more common and efficient while driving along long, straight highways.

Fig. 2.1: The first car equipped with a lane detector system [6].

After appearing for the first time, lane departure warning systems became a tool used in many different vehicles. In 2003, Honda introduced its Lane Keep Assist System (LKAS) on the Inspire model, based on images captured by a C-MOS camera mounted inside the front window, which consists of producing 80% of steering torque to keep the car in its lane [7]. The Japanese brand Toyota added a system to the Crown Majesta (Fig. 2.2), a model that includes an assist driver system. This was achieved by developing a technology that was able to interact with the driver in order to improve the direction of the vehicle during a trip. This was made possible by sending commands to the power steering system to encourage the driver to make a steering correction. In 2014, Tesla equipped the S model with an Advance Lane Assistance System [8]. This technology includes a feature that promotes a beeping and a steering wheel vibration when the vehicle changes lane without the driver signalling or turning the steering wheel.

Fig. 2.2: Toyota Crown Majesta [9].

2.2 Current Implementations

Nowadays, many car brands are developing Advanced Driver Assistance Systems (ADAS), which commonly rely on LKA (Lane Keeping Assist). This kind of technology tries to keep the car in the centre of the current lane if it starts to leave the lane. It is being achieved in intelligent vehicle systems based on classical techniques of computer vision such as the Hough Transform [10] and Canny Edge detector [11]. To process the road scenario in real-time, hardware such as Nvidia's Drive PX1 is used. This type of computer has as main characteristics: high accuracy and suitable time processing when used to compute systems based on neural networks. Tesla (Fig. 2.3) autopilot system is one of the technologies that uses this feature to make highway driving more enjoyable. Autopilot allows the car to make significant steering inputs and it has been updated since 2014. At this moment, the system has its processing power increased by 40 times when compared to the previous generation due to the adoption of a sophisticated computer developed by the company to run a neural network.

Fig. 2.3: Tesla model S equipped with the autopilot system [12].

Not only Tesla is developing this system but also Nissan (e.g. Q50 and Q60 models) with the Infiniti's Direct Adaptive Steering system and Mercedes-Benz which has been developing a Driver Assistance Package on the S-Class model. Volvo announced in 2017 [13] a semi-autonomous drive technology, Pilot Assist II. It includes a LKA system in the XC90 II model.

There are also several vehicles endowed with the capacity of proactively keeping the vehicle in the centre of the lane - LCA (Lane Centering Assist). Car companies like Audi, Honda [14] (using a windshield-mounted camera to look for lane markers, and the Electric Power Steering to help steer the vehicle) and Hyundai equip some of their models with this type of technology.

2.3 Related Work Developed at LAR

There is a work at LAR authored by Morais [15], who developed and implemented two different methods to detect road lines – "Road Time Detection of Lane Markers in Urban Streets" and "The Lane Tracker". The first, described by Mohamed Aly [16] in 2008, is divided into several phases:

The first one consists of the IPM technique (Inverse Perspective Mapping or "Bird's Eye View"), which is related to the application of a transformation on the image perspective (Fig. 2.4). This technique enables to obtain the representation of parallel line roads and not convergent.

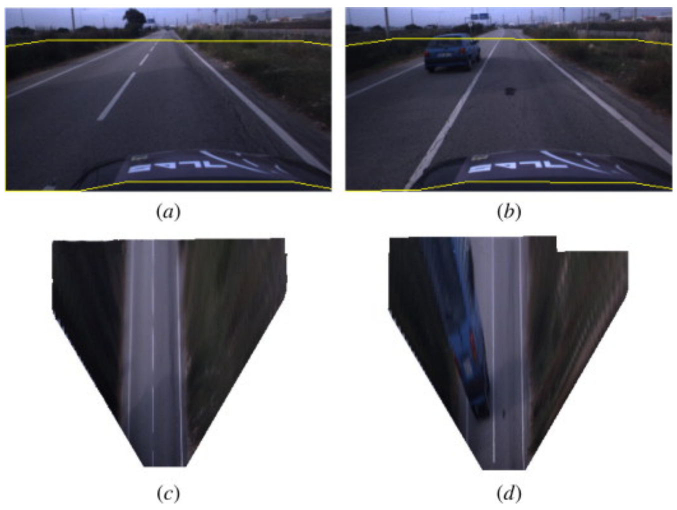


Fig. 2.4: Two initial images (a) e (b) and the respective IPM applications (c) e (d). The yellow line delimits the used area to get the IPM projection [18].

In the next phase, entitled "Filtering and Thresholding", the image is processed through a Gauss 2D filter and a threshold is applied in order to obtain a result similar to the one presented in Fig. 2.5-Right.

Thus, a simplified version of the Hough Transform is applied, followed by the application of RANSAC Line Fitting [17], which increases the detection robustness. Then it's crucial to hone the method to achieve a better lane detection and for that, the RANSAC Spline fitting is applied which fits the obtained lanes to a spline.

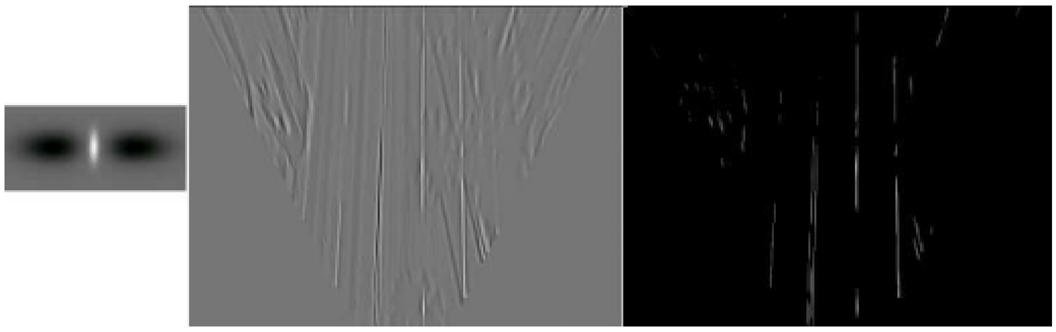


Fig. 2.5: Processed Image. Left: Kernel used in filtering phase. Centre: Image after kernel application. Right: Image after the threshold application [16].

The second algorithm implemented by Morais is based on a Matlab® toolbox that has the Hough Transform as the basis of the algorithm. In an introductory phase of this method, the image is divided into two regions, although only the bottom part is used for searching edges. The lanes found are stored to use as a point of comparison with the next frames.

2.4 Related Work Developed in Other Contexts

In 2018, Chuan-en-Li [19] proposed an algorithm constituted by classic visual computing techniques (Fig. 2.6). This approach uses the Hough Transform as the main technique in order to extract the road lines, yet, a Canny detector (a multi-stage technique) is applied first. This method was designed to be an optimal edge detector in a grayscaled frame. The next step of the algorithm is to segment the lane area through a handcrafted mask that is defined by the coordinates of three image points. It allows focussing on the important areas on the next stage of the algorithm. After that, the Hough Transform is applied, which transforms the Cartesian Space in the Polar Space. So, the lines (in Cartesian coordinates) are represented as points and a line on this last system of coordinates corresponds to a set of lines. Thus, if the number of intersections in the polar coordinate system exceeds a defined threshold, this intersection (represented by θ and r parameters) is considered as a line.

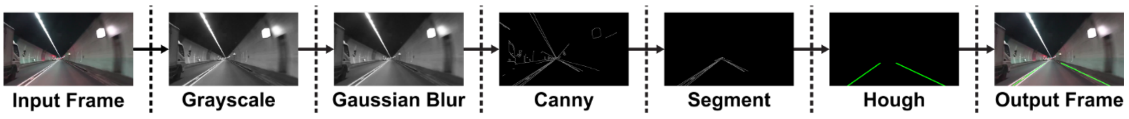


Fig. 2.6: The Chuan-en-Li algorithm pipeline [19].

In [20], several authors developed a road lane lines detection method using advanced computer vision techniques during the Udacity's Self-driving Engineer Nanodegree program. All the algorithms implemented have a similar pipeline. The method starts by applying a perspective transformation to the input frame, which means that the initial vehicle perspective given by the camera becomes a perspective of a bird in the sky (as explained in Section 2.3). After that, colour thresholds are applied to the HSL colour space, where H (Hue) corresponds to the dominant wavelength of the colour, S (Saturation) represents the relative purity of a colour (i.e. a colour without any white is fully saturated) and L (Luminance) quantifies the amount of white present in an image. In this stage, some authors also apply the colour threshold to the RGB colour space. Then, a Sobel operator is applied to the lightness channel of the image to eliminate the pixels with weak changes of lightness. After that, the algorithm fits a curve for each line with a second-degree polynomial function to get the radius of the curvature of each lane. Finally, the information obtained from the sky-view is transformed into the vehicle view to project the road boundaries prediction in a real frame that comes from the camera (Fig. 2.7).

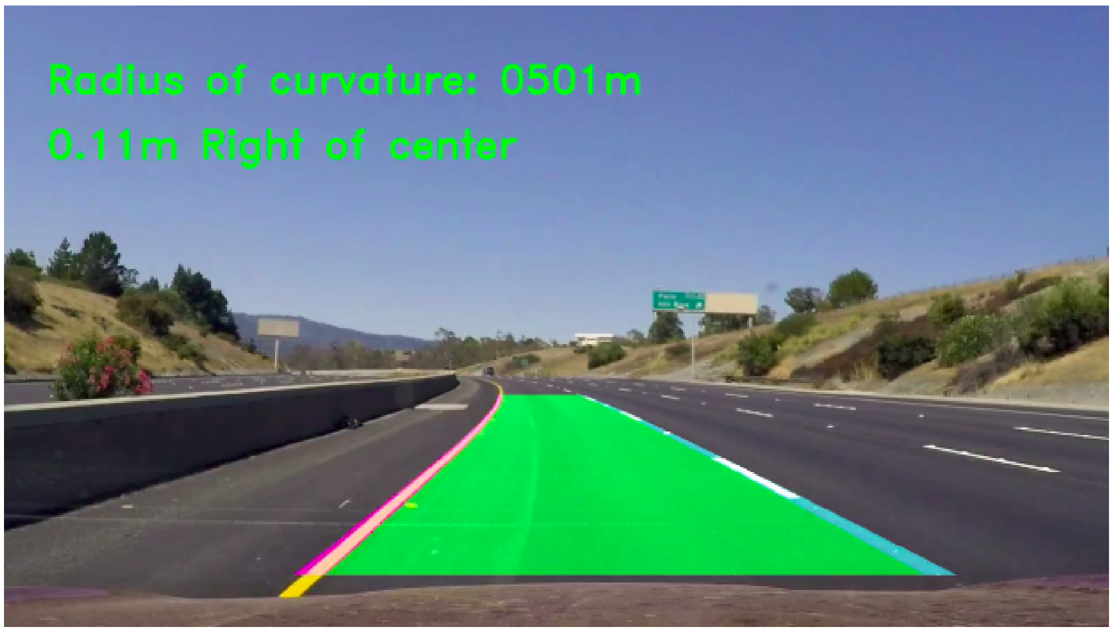


Fig. 2.7: Result obtained from an application of Udacity's algorithm [20].

A different strategy was applied in [21], which contemplates the use of deformable templates to locate lane boundaries without using edge detectors. In [22], a B-Snake based lane detection and tracking algorithm is developed. The key of this work is the B-snake technique since it has practical and computational advantages against other lane models (e.g. B-Spline) such as a greater description of more complex road shapes, more robustness when it faces complex environments (Fig. 2.8), less processing time and it is more suitable for lane tracking application.

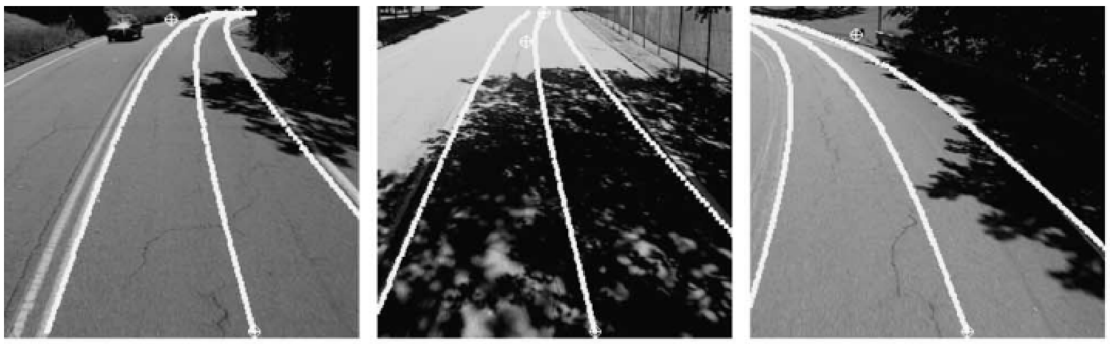


Fig. 2.8: Result obtained from the B-Snake application method [22].

In [23], the authors propose a procedure of several stages (Fig. 2.9) for road lane detection. First, the Canny edge detector is applied to acquire an ”edge map” from the road image. After that, the edges that were found are labelled according to the type of line that they represent. This is achieved through a comparison between the edge map and several different types of templates that are related to each type of existing lines. Now, the lines that do not correspond to the road lines are suppressed by applying a priority and orientation based searching method to the previous labelled lines. The next step of the procedure is to strengthen the confidence of a hypothetical road lane through a linking condition. Finally, the K-mean clustering algorithm is computed to find the localisation of the road lines in the image.

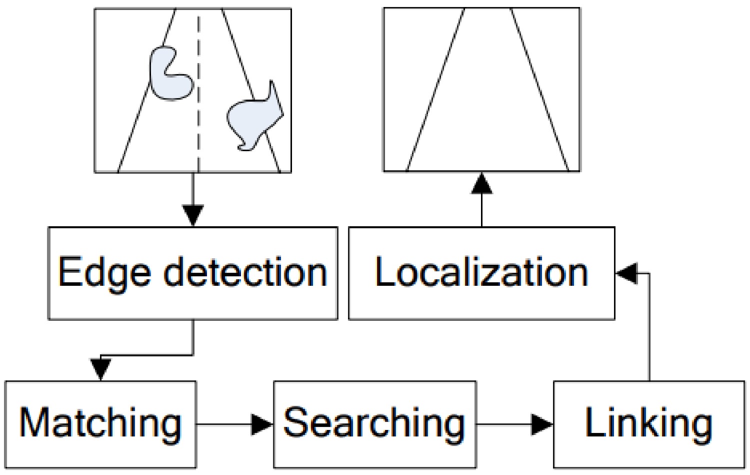


Fig. 2.9: Pipeline of the algorithm that uses several templates for different types of lines [23].

In terms of existing ROS packages, there is one developed by Nicolas Acero [24], who also used classic techniques such as the IPM, application of thresholds and the Kalman Filter, which is a filter that provides the lanes tracking based on previous frames. This filter allows the road lane recognition in a frame where they are not represented for some reason.

A group of authors identifies the solution to the problem as consisting of a multi-feature fusion [25]. First, it proposes an edge lane marking segmentation, which starts by extracting the image foreground through the Red channel based on the Otsu's Method. The threshold applied in this phase (FBS-Foreground/Background Segmentation) is determined through a dataset created previously. After that, the probability density functions of a normal distribution are calculated for the channels Red, Green and Blue from the RGB colour space and the saturation channel from the HSV colour space. The images used for these calculations are included in a training dataset. Thus, the probability of each pixel belongs to a lane region (P_{pixel}) is provided by the product of the probability of the pixel belongs to the lane region according to each channel already mentioned (Pr_{pixel}, Pg_{pixel}, Pb_{pixel} and Ph_{pixel})-given by Eq. 2.1:

$$P_{pixel} = Pr_{pixel} \times Pg_{pixel} \times Pb_{pixel} \times Ph_{pixel} \tag{2.1}$$

Therefore, a binary probabilistic image is computed through a threshold applied to all the P_{pixel} (Fig. 2.10 (c)). After that, a Canny filter is applied to the FBS result (Fig. 2.10 (b)) in order to extract a set of points which are located within the road lane (Fig. 2.10 (d)). Then, this set of points are intersected with the binary probabilistic image to create a candidate set of road lane points. Finally, the lane points are clustered by using a distance between pixels method (Fig. 2.10 (e)) and fitting by a RANSAC model (Fig. 2.10 (f)).

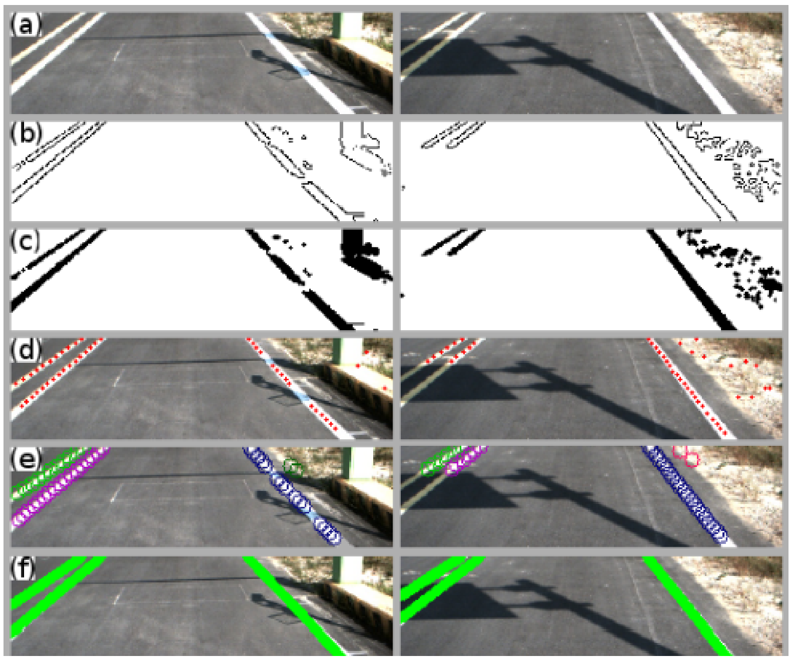


Fig. 2.10: Lane marking feature extraction experimental results. (a)Input image. (b) Edge detection results after applying Canny edge detector on the FBS image results. (c) Binary probabilistic image results. (d) Middle point extraction. (e) Lane marking clustering results. (f) Fitting model results. [25].

There are also other types of methods, which are based on learning related to AI. AI is defined as a panoply of abilities such as think, learn, adapt and react that are given to a computer/machine. Directly related to Artificial Intelligence is Machine Learning, whose main characteristic is the improvement of an algorithm's performance as the train through more data is increasing over time. Finally, it is important to clarify what is Deep Learning because it is strongly connected with the two concepts previously mentioned. It is a technique that is being developed to increase the efficiency and robustness of this type of algorithms. Technically, it is a subset of Machine Learning in which multilayered neural networks learn from a vast amount of data to get knowledge to later return the output. Besides being the most well-known method, Deep Learning is also the most complex technique because it is based on several different areas as biology, since it uses the human learning as the basis of the learning that is provided to the machine. Concerning this type of method, Chuan-en-Li also developed a method that consists of a Spatial Convolutional Neural Network (SCNN), which is a method that returns reasonable results when the lighting conditions are poor and the road is unstructured. This type of neural network is efficient in getting spatial relations because the layers of the neural networks are divided into slices to increment the image spatial data (Fig. 2.11).

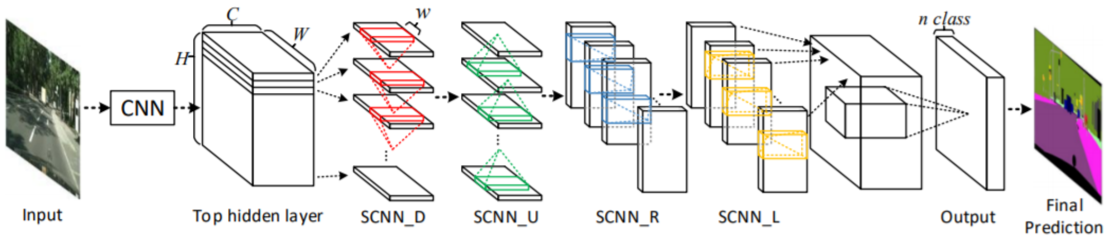


Fig. 2.11: SCNN's architecture [19]. This architecture is composed by a set of slices, in which each one, sequentially, passes information to the succeeding slice only after it has received information from the preceding slices), allowing message passing of pixel information between neurons within the same layer, effectively increasing emphasis on spatial information.

Chapter 3

Experimental Infrastructure

This chapter is divided into two main sections. Section 3.1 presents and describes the hardware used as well as the hardware developed during the dissertation. In Section 3.2, all software used to get the final result is enumerated and described.

3.1 Hardware Used and Designed

3.1.1 Point Grey FL3-GE-28S4-C

The Flea3 FL3-GE-28S4-C (Fig. 3.1) is a 2.8 Megapixel colour GigE Vision digital camera based on the Sony ICX687 EXview HAD CCD II image sensor. The camera runs at 15 fps at full resolution (1928 × 1448), measures 29 × 29 × 30 mm, and weighs 38 g. Table 3.1 details the camera's most relevant specifications.

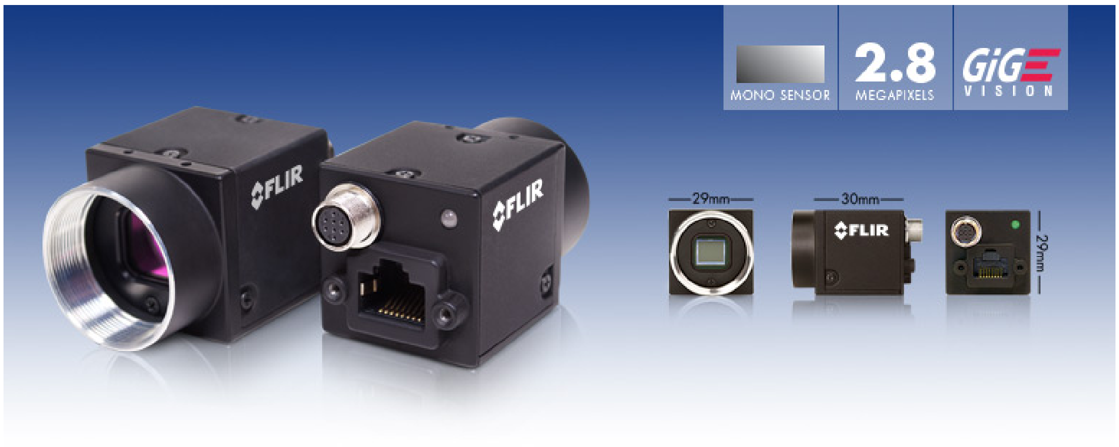


Fig. 3.1: Point Grey FL3-GE-28S4-C.

The sensor has several pixel formats that can be chosen: Raw (image data unprocessed), Mono (image data is monochrome), RGB (each pixel represents three intensities-red, green and blue) and YUV, that assigns both brightness (Y) and colour (UV) values to each pixel. Other variants that can be considered include the number of bits per pixel that can be sent to form the image, which may vary between 8 and 24 bits per pixel. The pixel format chosen was Raw 8.

Finally, this camera is capable of running in different video modes, all of which allow the user to select a specific ROI (region of interest) of the image. Also, some modes can increase the number of frames per second by aggregating pixels. This process is called `binning`, whose main characteristic is to join pixels in two possible directions (horizontal or vertical) to reduce the effective image resolution and, consecutively, increase the frame rate. There is, also, a process known as `subsampling` or `decimination` that skips every second pixel horizontally and vertically (2×2) as shown in Fig. 3.2.

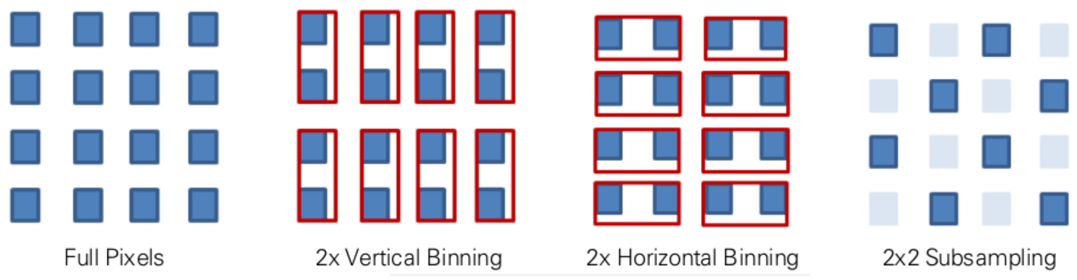


Fig. 3.2: Aggregation and decimation of pixels.

With these techniques, the image quality may be poorer since it reduces the effective image resolution, so it is necessary to be aware of this. The video mode used is Mode 1 because increases the velocity of image formation by performing a $2\times$ vertical `binning` and $2\times$ horizontal `subsampling`. This can be crucial in getting up-to-date information about the real scenario.

Table 3.1: Point Grey FL3-GE-28S4-C specifications [26]

Max. Framerate (FPS)	15 @ 1928×1448
Max. Resolution	1928×1448
Imaging Sensor	Sony ICX687 1/1.8" CCD
Version	Colour/Mono
Weight (g)	38
Dimensions (mm $\times$ mm $\times$ mm)	$29 \times 29 \times 30$

3.1.2 Camera Setup on ATLASCAR2

Before assembling the camera in ATLASCAR2, the location to install the camera in the vehicle was studied. There were two valid possibilities: inside the car or on the car roof. Inside the car, the camera would be installed behind the windshield above the rear-view mirror, similar to the cameras installed in the Autopilot system from Tesla (Fig. 3.3).

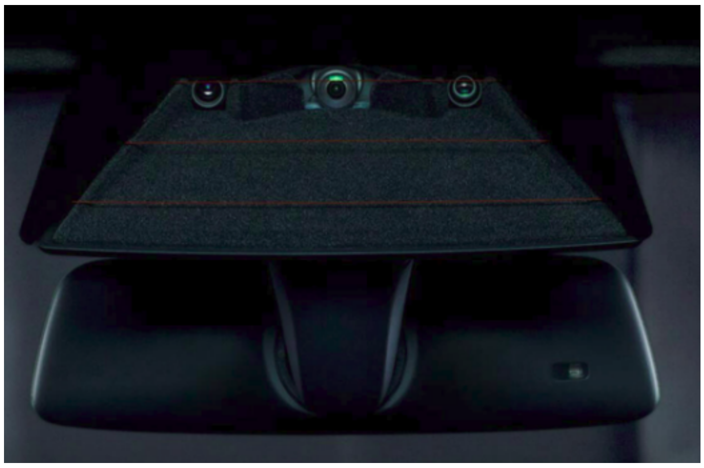


Fig. 3.3: Camera's location in a Tesla.

The main advantage of this is that the camera is protected from the outside agents but the practical installation would be complicated because of the cameras' dimensions and the cables that have to be connected to the sensor.

So, the placement selected to settle the camera was the car roof (Fig. 3.4), which enables to install the camera and manage the camera's pitch angle easier than the possibility mentioned before.

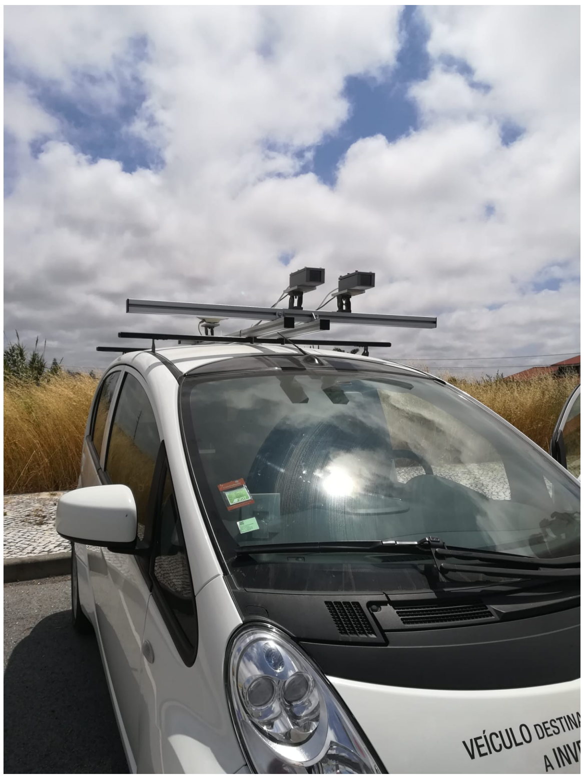


Fig. 3.4: Cameras location in ATLASCAR2.

Regarding the camera assembling on ATLASCAR2, it is quite important to establish some goals to the final infrastructure:

- Protective - a protection box to the camera must be contemplated to protect it from dust and rain;
- Rigid - the camera support has to be made in a rigid material to prevent camera movements;
- Easy to assemble and disassemble;
- Watertightness - the box that protects the camera has to prevent the liquids inlet.

Based on the previous specifications, a box (Fig. 3.5) was designed to protect the camera from the outside agents. This box was 3D printed and assembled with screws M5. Also, a plate (Fig. 3.6) was designed to fix the box to the rotative support. This support is attached to two Bosch aluminium profiles with 1.5 meters long, which is assembled to the car roof. Finally, gutters were used to protect the Ethernet cable to the computer connection and the power cable (8-pin GPIO connector) to the UPS (Uninterruptible Power Supply) connection. Fig. 3.7 shows the final setup of ATLASCAR2.

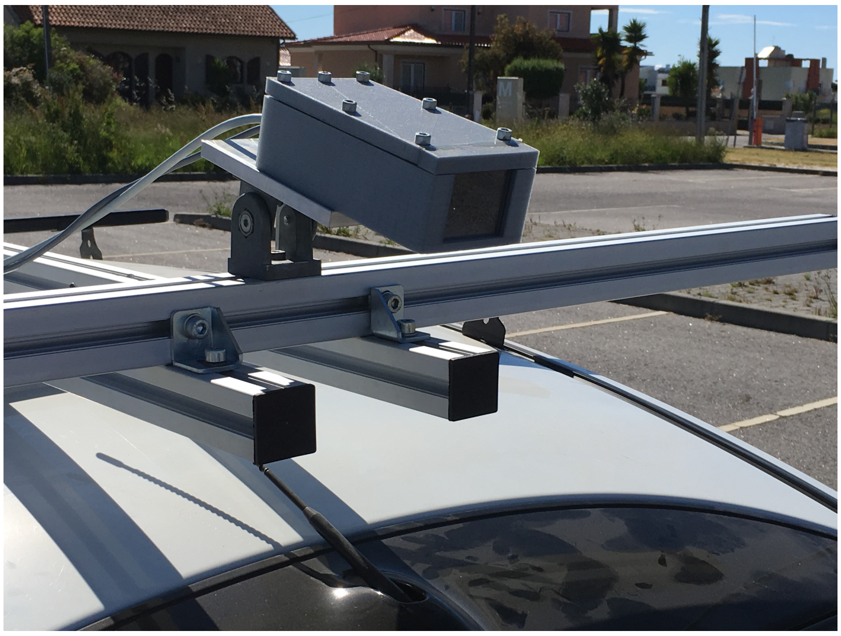


Fig. 3.5: Box that was created to protect the camera (Appendices B and C).

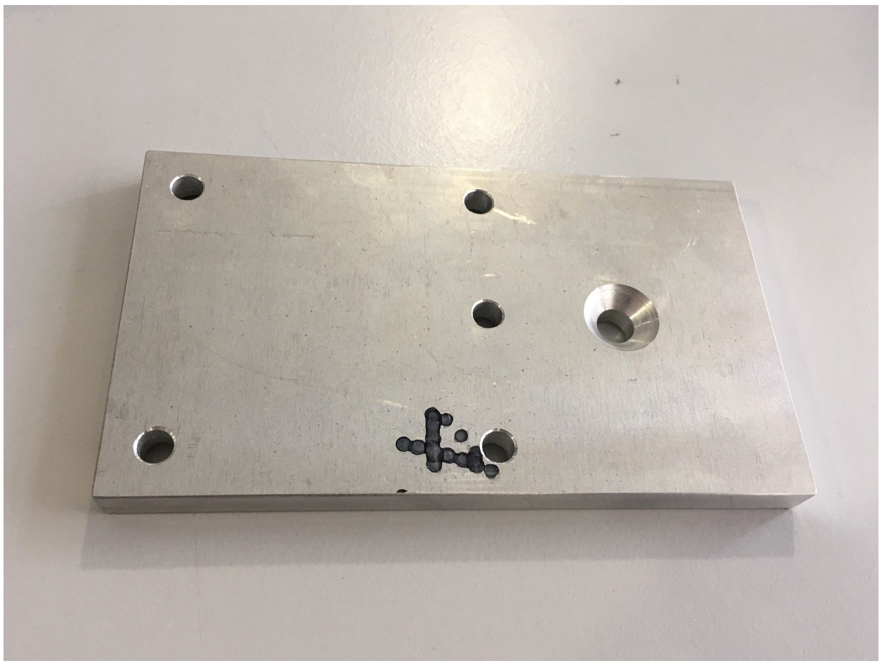


Fig. 3.6: Plate to fix the box to the rotative support (Appendix A)
.

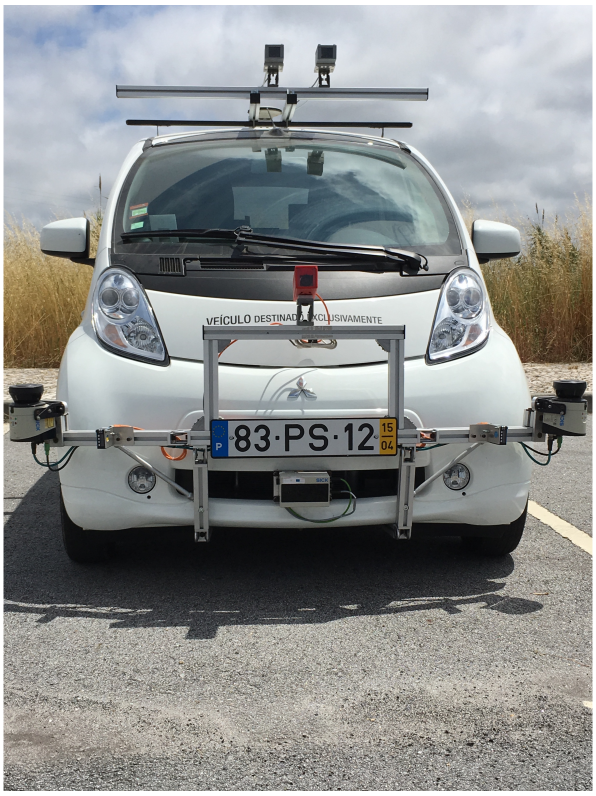


Fig. 3.7: Final ATLASCAR2 setup.

3.2 Software Used

3.2.1 ROS - Robot Operating System

ROS (Robot Operating System) is an open-source framework and a meta operating system that operates on top of one OS (Operating System) like Linux. It enables to manage and control hardware such as sensors, cameras and robots due to a well-planned communication protocol. This communication protocol is based on a publisher-subscriber logic. In a deeper view, ROS has UNIX processes called nodes that can communicate with each other. Each node can represent one sensor or a specific task like processing images or processing a point cloud that is provided by a sensor. These nodes communicate through messages that are published in topics. These messages can be "ROS standard" or customised by the user. On the other hand, the nodes can communicate through another type of architecture - ROS services. They differ from the regular communication between nodes because, in this case, a request is asked and a response is returned.

A ROS project is controlled and dependent on a ROS Master because it allows the communication between nodes and keeps a registry of all the nodes. This also provides a database of parameters that can be accessed and used in different locations.

The ROS distribution used in this project is ROS Melodic.

Rqt Package

Rqt hosts several plugins for displaying ROS information such as `rqt_console` that allows to filter messages by various means (e.g. which node publishes them), `rqt_logger_level`, which helps the user to do debugging because it shows messages marked by different levels (debug, info, warn, error or fatal) and `rqt_graph`, that shows on a visual way, the type of existing connections in a ROS architecture. In this project, this software framework made it easier to show the results because it allows displaying images in the same window published in different topics (Fig. 3.8).

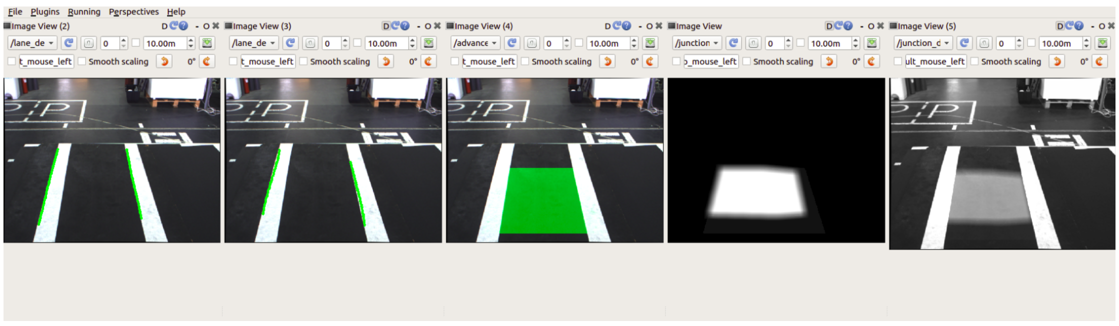
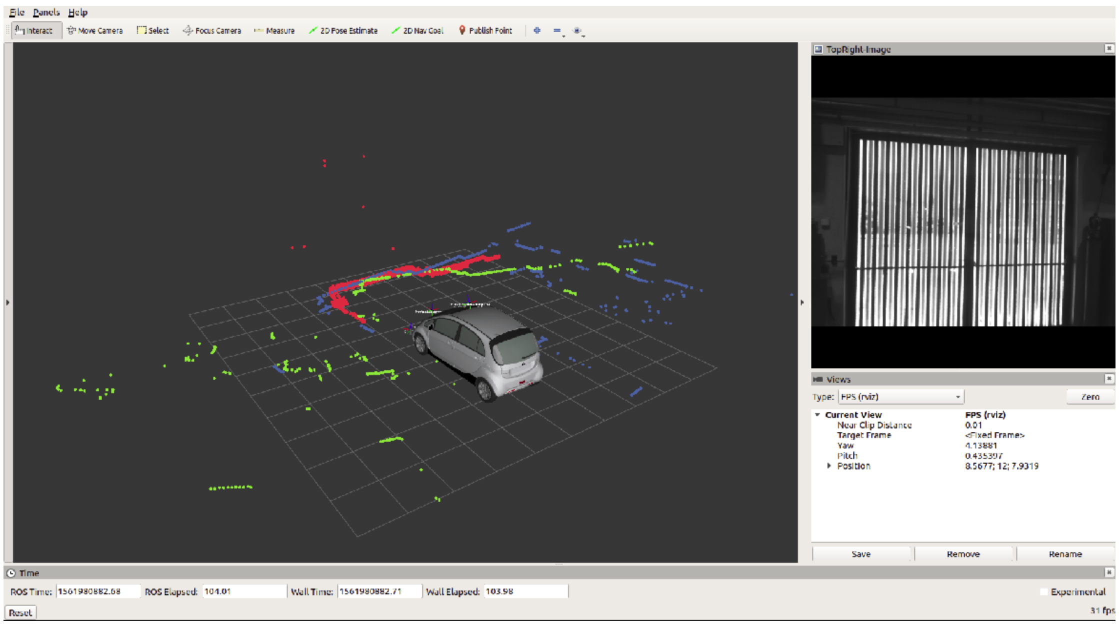


Fig. 3.8: One of the examples of using rqt in this project.

Rviz

Rviz is an abbreviation for "ROS visualisation" [27], and its main function is to show the visual information of a ROS project such as the robot model, images captured from a camera, of point clouds generated by data that comes from a sensor. The node launched by the Rviz package subscribes to the available topics to display the information included on them. It also provides a grid of a ground plan, which was quite important in this project, because it provides the visualisation of the sensors data included on the ATLASCAR2 setup (Fig. 3.9).



Type: FPS (rviz)

Current View	FPS (rviz)
Near Clip Distance	0.01
Target Frame	<Fixed Frame>
Yaw	4.13881
Pitch	0.435397
Position	8.5677; 12; 7.9319

ROS Time: 1561980882.68 ROS Elapsed: 104.01 Wall Time: 1561980882.71 Wall Elapsed: 103.98

Fig. 3.9: Rviz Interface. Through this interface, the sensors data can be visualised in real-time.

3.2.2 ROS Packages used in this Project

`pointgrey_camera_driver`

The `pointgrey_camera_driver` ROS package is specially designed for Pointgrey cameras and includes many interesting features such as the camera "urdf" and a configuration file that allows to change the camera details (video mode e.g.) without accessing to the "Flycap" application.

`camera_calibration`

The `camera_calibration` package enables to calibrate monocular or stereo cameras using a checkerboard calibration target based on OpenCV camera calibration tool. It returns a file, which includes the intrinsic camera matrix, a distortion vector and the rectification and projection matrices.These calibration parameters are used as inputs of the `image_proc` package.

`image_proc`

The `image_proc` package rectifies the image captured from a camera. This is possible because the package contains a node that removes the image distortion through the camera parameters.

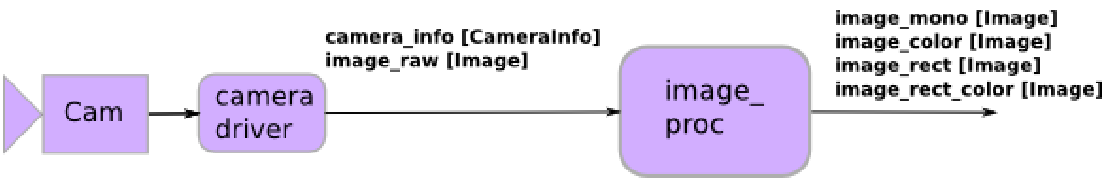


Fig. 3.10: `image_proc` ROS package architecture [28].

3.2.3 Other Software Tools

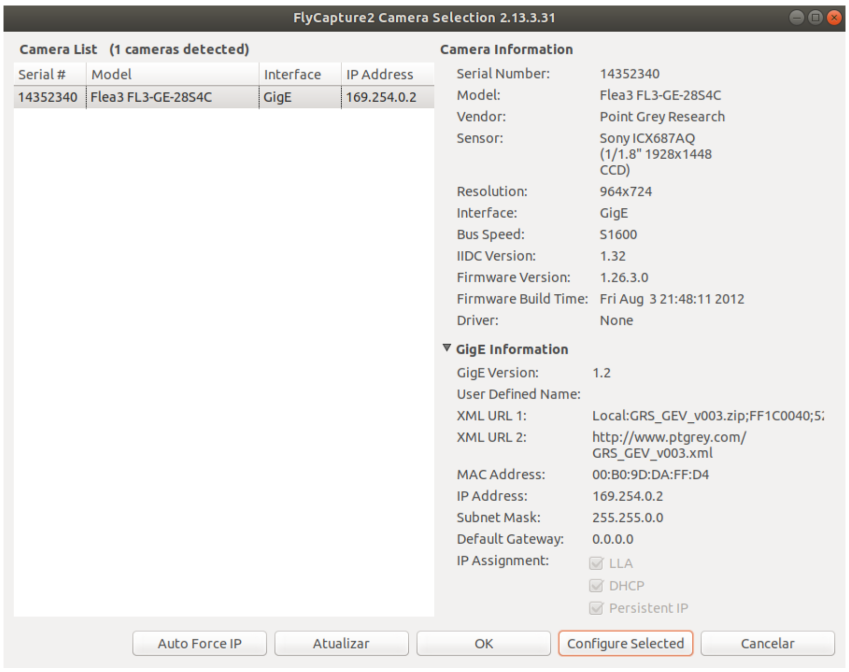
OpenCV

OpenCV is an open-source computer vision library that is written in C and C++ and runs on Linux, Windows, MacOS, Android and iOS [29]. It can be used in several programming languages like Python, Ruby, Matlab and Java. It contains vision functions to process images and some basic machine learning algorithms to help people to build fairly sophisticated vision applications with high computational efficiency. The library is subdivided in modules with different functionalities such as camera calibration (`calib3d`), image processing (`imgproc`) or video analysis.

This project is dependent on this tool since the main aim of the project is: visual perception of the road detecting the road boundaries.

FlyCap

The FlyCap application (Fig. 3.11) is a generic, easy-to-use streaming image viewer included with the FlyCapture SDK [30] that can be used to test many of the camera capabilities. It enables to view a live video stream from the camera, save individual images, adjust the various video formats, frame rates, properties and settings of the camera, and access camera registers directly.


FlyCapture2 Camera Selection 2.13.3.31

Camera List (1 cameras detected)

Serial #	Model	Interface	IP Address
14352340	Flea3 FL3-GE-28S4C	GigE	169.254.0.2

Camera Information

Serial Number:	14352340
Model:	Flea3 FL3-GE-28S4C
Vendor:	Point Grey Research
Sensor:	Sony ICX687AQ (1/1.8" 1928x1448 CCD)
Resolution:	964x724
Interface:	GigE
Bus Speed:	S1600
IIDC Version:	1.32
Firmware Version:	1.26.3.0
Firmware Build Time:	Fri Aug 3 21:48:11 2012
Driver:	None

GigE Information

GigE Version:	1.2
User Defined Name:	
XML URL 1:	Local:GRS_GEV_v003.zip;FF1C0040;5
XML URL 2:	http://www.ptgrey.com/ GRS_GEV_v003.xml
MAC Address:	00:B0:9D:DA:FF:D4
IP Address:	169.254.0.2
Subnet Mask:	255.255.0.0
Default Gateway:	0.0.0.0
IP Assignment:	✓ LLA ✓ DHCP ✓ Persistent IP

Fig. 3.11: Flycap Interface.

SolidWorks

SolidWorks is a CAD (Computer-Aided Design) software that allows to model parts easily and cost-effectively. This software includes a user-friendly GUI (Graphical User Interface) as compared with other CAD solid modelling software.

In this project, this software was determinant when was necessary to create parts to fix the camera to the vehicle as was showed in Section 3.1.2.

3.3 Summary

This project is based on the software tools and the hardware parts that were explained and described in this chapter. ROS is the connection link between the main hardware (camera) to the software (OpenCV) since it publishes images that are captured by the cameras, that will be processed later by OpenCV to obtain the final result of the developed ROS architecture.

Intentionally blank page.

Chapter 4

Calibration

This chapter describes the calibration methods that were implemented to undistort the original image that is captured from the camera (intrinsic calibration) and find the transformation between the coordinate frames belonging to the system (extrinsic calibration).

4.1 Intrinsic Calibration

The process of doing the intrinsic calibration consists of determining the intrinsic camera parameters: the focal length in two axis (f_x and f_y), the optical centers (c_x and c_y) and the skew coefficient (s). In addition to undistort the original camera image, this process relates pixel coordinates with metrics coordinates.

The matrix that combines all the camera parameters is the camera matrix (Eq. 4.1):

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

As introduced in Section 3.2.2, the intrinsic camera calibration was made through the `camera_calibration` ROS package [31], which by using a checkerboard, determines the focal length and optical centers parameters, not taking into account the skew factor. This calibration technique is based on the OpenCV camera calibration method that uses the pinhole camera model. The main objective of this model is to find the relations between the coordinates of a point in a 3D space and its projection onto the image plane.

The interface (Fig. 4.1) provided by the package allows the user to know if the checkerboard has already occupied all the positions (left, right, bottom and top) in the field of view of the camera. A successful calibration gives rise to a "`yaml`" file that contains the camera matrix, the distortion vector, and the rectification and projection matrices.

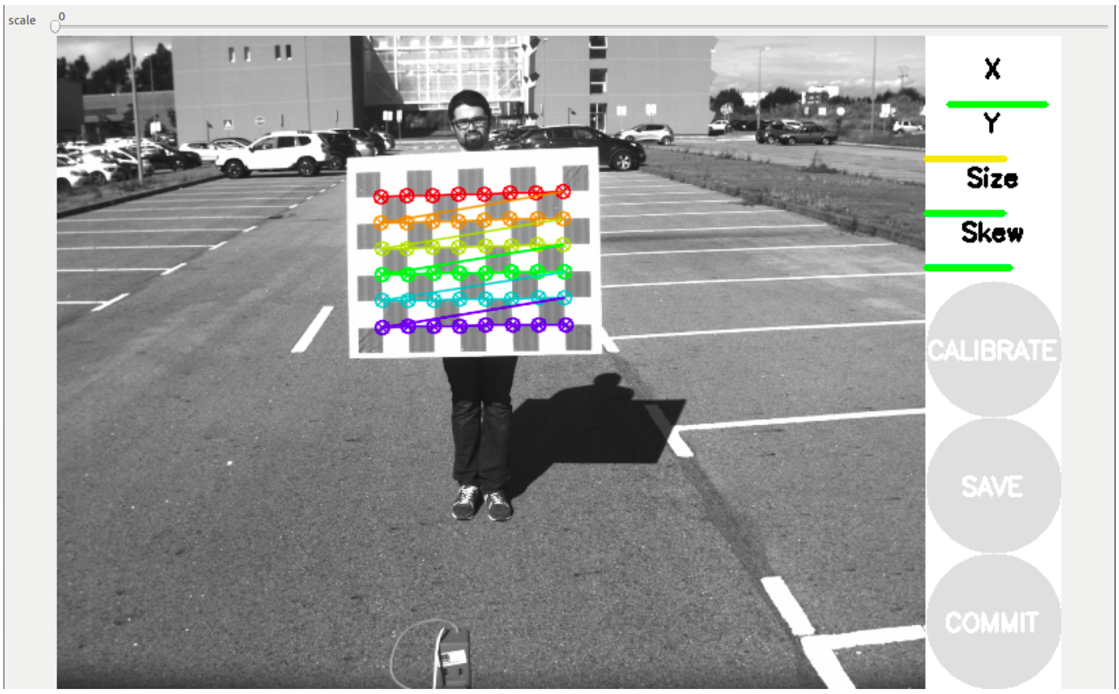


Fig. 4.1: `camera_calibration` interface.

Image Rectification

After determining the intrinsic parameters, it is everything set to acquire the undistort image. This process was achieved by using the `image_proc` ROS package. It is based on a node that subscribes to two topics: the `camera_info` topic, which has all the intrinsic camera parameters and the `image_raw` topic that contains the image captured by the camera. This node transforms the original image into a rectified image (with the distortion corrected). Therefore, the pixels of the image undergo the transformation resulting from the camera matrix (K) following the distortion vector (D). If the image had to be rotated then a rotation matrix would have to be applied (this is an optional step). Finally, the main node of the method publishes the rectified image and the non-rectified image.

4.2 Extrinsic Calibration

In order to analyse data in a common frame, knowledge of the geometric transformations between each camera and the reference frame is required. The frame that is considered as the reference is named "*moving_axis*" and is represented in Fig. (Fig. 4.2). The referential used to perceive the road boundaries based on LIDAR is also "*moving_axis*". An interesting study that can be developed later is the data fusion between the road boundaries detected from two different sources: LIDAR sensors and cameras, that is why the frame chosen as the reference is the same. In conclusion, the frame that is going to be used as a reference to the road boundaries detection through visual perception is the same as the one used in LIDAR detection.

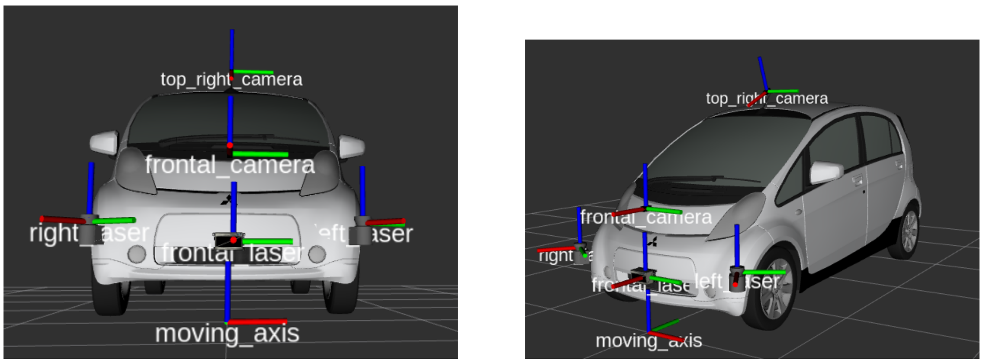


Fig. 4.2: Frames representation on the ATLASCAR2.

Regarding the transformation that should be known, the graph presented in Fig. 4.3 illustrates the geometric transformations inherent to the problem.

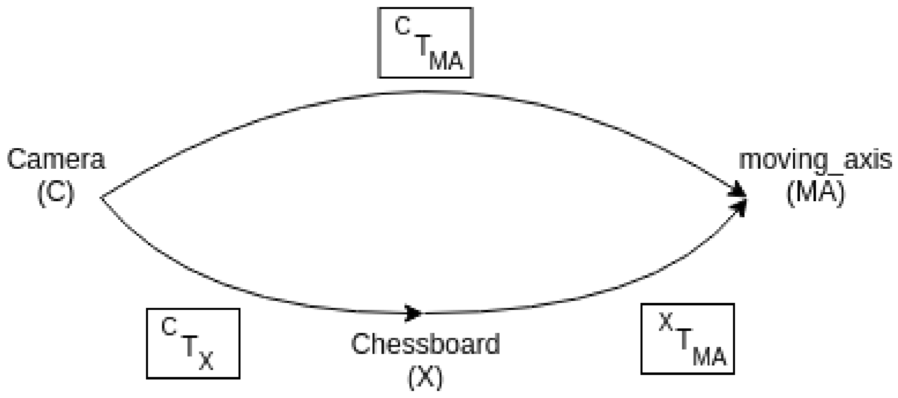


Fig. 4.3: Geometric transformations associated to the extrinsic calibration problem.

The Eq. 4.2 is derived from the graph transformation.

$$^{C}T_{MA} = ^{C}T_{X} \cdot ^{X}T_{MA} \tag{4.2}$$

Where $^{C}T_{MA}$ (unknown variable) is the transformation between the camera and the *moving_axis* frame, $^{C}T_{X}$ is the transformation between the camera and the chessboard, and $^{X}T_{MA}$ is the transformation between the chessboard and the *moving_axis* frame.

The procedure was implemented in Matlab® environment, through the application `Camera Calibrator`. In this application, one image of a chessboard on the road is captured (Fig. 4.4).

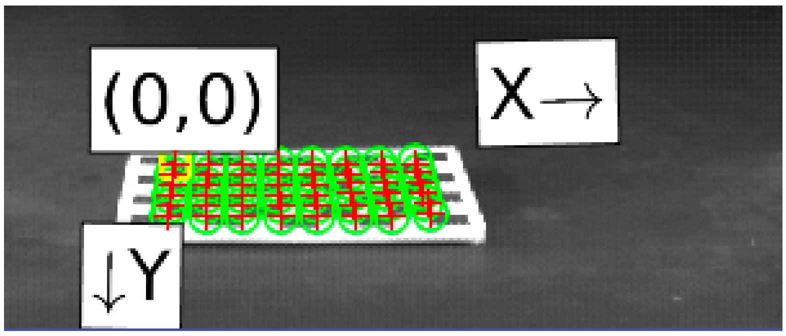


Fig. 4.4: Detection of chessboard points through `Camera Calibrator` application.

Then, the application returns the coordinates of the image in pixels and millimetres, since the dimensions of the chessboard are given at the beginning of this stage. After that, the function `extrinsics` was computed, to find the translation vector and the rotation matrix of the transformation between the camera and the chessboard origin. As inputs this Matlab® function has the localisation of the chessboard points in pixels and millimetres and the intrinsic camera parameters. These last parameters were achieved in section 4.1.

Since the transformation between the camera and the chessboard is determined, now the transformation between the reference frame and the chessboard is determined. This is the step more inaccurate of the deployed method, however, there is no other possibility at this moment to do this type of calibration on ATLASCAR2. Therefore, the origin of the chessboard detected in the last step (transformation camera-chessboard) was already aligned with the reference frame to minimise the error caused by the differences in the angles. At this point, the transformation between the reference frame and the chessboard is measured through a laser range finder (Fig. 4.5).

Fig. 4.5: Length measuring tool used to find the transformation between the reference frame and the chessboard.

Finally, all the transformations are known then the transformation between the camera and the reference frame is calculated.

Intentionally blank page.

Chapter 5

Multi-Camera and Multi-Algorithm Architecture

This chapter presents the architecture developed to combine or fuse the results of each algorithm (representation of the road boundaries). The main goal is to develop a multi-camera architecture capable of gathering information from multiple algorithms and then use it to return the road boundaries with more accuracy.

This chapter describes the ROS-based architecture, the influence of `time stamp` preservation on the images throughout the architecture, and the nodes/topics used. Finally, the image processors packages are summarised and the main launch files are detailed.

5.1 The Base Architecture

In order to increase the quality and quantity of the information obtained from the road boundaries detection, a ROS-based architecture was developed. It starts from a rectified image provided by the `image_proc` package. This image is the input of the algorithms, which return the road lane lines. In this stage, the architecture is prepared to receive results from two different types of algorithms (Fig. 5.1): the spatial localisation of the road lanes in the image, or the polygon that represents the road zone.

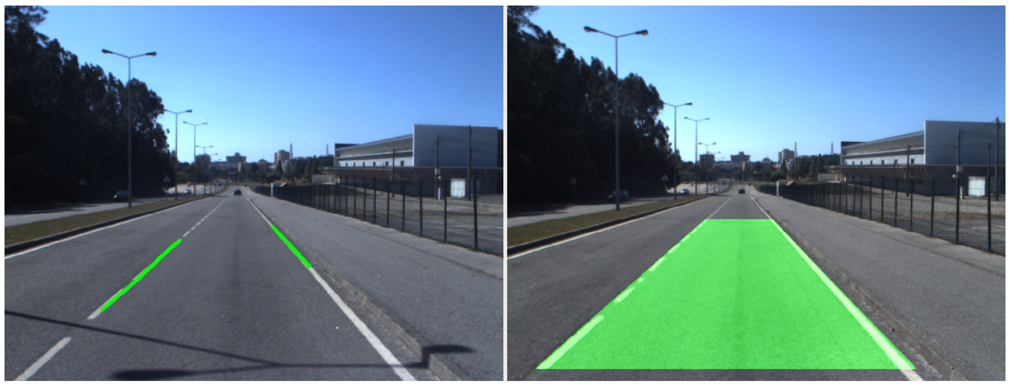


Fig. 5.1: The different types of data that can be processed by the architecture. Left: Spatial localisation of the road lines. Right: Polygon that represents the road zone.

If an algorithm returns the spatial localisation of the road lines, then the architecture processes this information, turning it into a polygon. After that, the polygon is merged with the data that comes from the algorithms that provide the polygon representation as a road zone. Finally, after every algorithm result is in the same data type (polygons), a confidence road map is built. Fig. 5.2 shows a flowchart that represents the general behaviour of the architecture when facing these two types of algorithms.

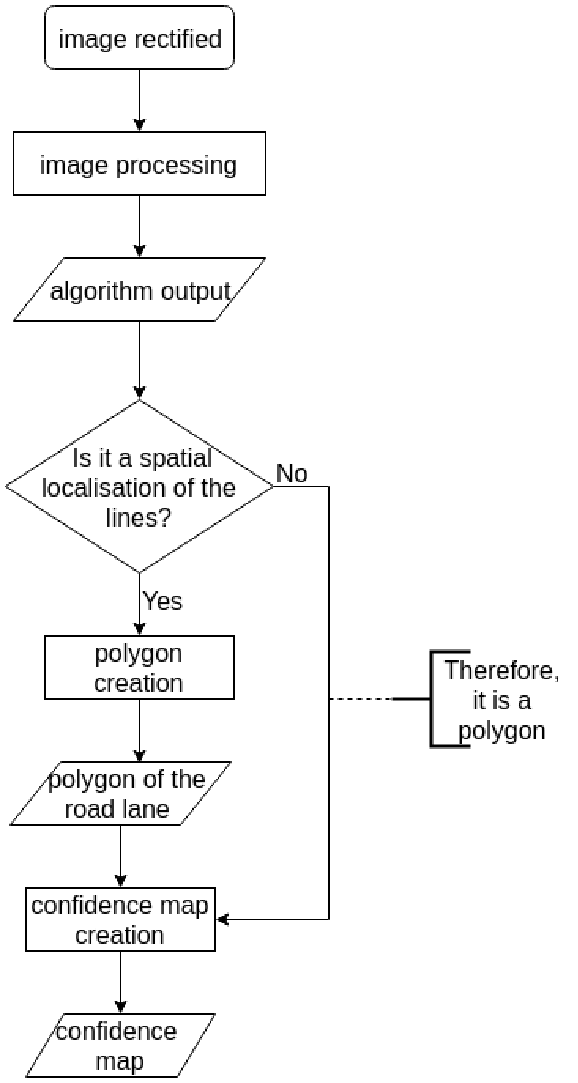


Fig. 5.2: The generic behaviour of the architecture for each type of algorithm output. From the top to the bottom: the rectified image is processed (image processing) by an algorithm that returns or a polygon or the spatial localisation of road lines (algorithm output). If it is the spatial localisation of the road lines in the image, then it passes through an intermediate stage (polygon creation) to build the polygon from those road lane lines coordinates. If the output of the processor algorithm is already a polygon, it proceeds directly to the construction phase of the confidence map.

5.1.1 Nodes and Topics Presentation

In terms of ROS computation, this project can be broken down in three main stages: 1) Rectified image determination; 2) Image processing; 3) Data combination. Each stage takes into consideration the source of each topic, hence, `namespaces` are used to keep the architecture understandable and clean (e.g. the topics that are published by the image processor node `lane_detector_node` have as `namespace` "`lane_detector`"). At this point, all nodes and topics are explained, not considering the case of multi-cameras for the exposure of the architecture to become clearer. The multi-camera approach is detailed in the last topic of this sub-section.

Rectified image determination

In this stage, the most important node is the one that is conceived by the `image_proc` package. Its name is the same as the package, the inputs are the topics `/camera_info` and `/image_raw`. The former is the one that contains the information which concerns the intrinsic camera parameters, and the latter contains the image that is captured by the camera. The `image_proc` node publishes the image rectified in the `/image_rect_color` topic (Fig. 5.3).

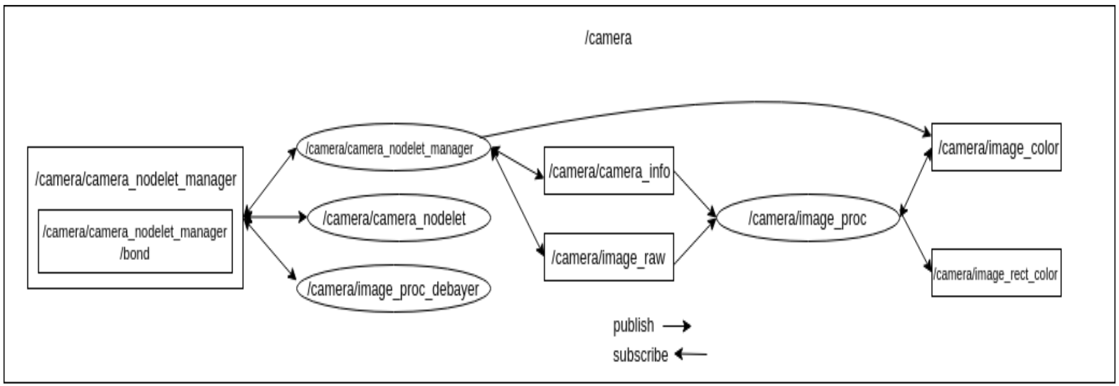


Fig. 5.3: Part of the computation graph to one camera that corresponds to the rectified image determination.

Image Processing

After acquiring the rectified image, two algorithms are applied (they are described in detail in a later section). The nodes that subscribe to the rectified image and publish the algorithms' outputs are named `lane_detector_node` and `advanced_algorithm_node`. Since the `lane_detector_node` has interesting parameters (to manipulate, which are related to the computer vision techniques that are computed in the algorithm), a clone of this node was made with different parameters. It implies that, instead of two processor nodes, there are three (`lane_detector_node`, `lane_detector2_node` and `advanced-_lgorithm_node`). These nodes somehow publish the road lane lines or the road zone in the respective ROS topics: `/lane_detector/lane`, `/lane_detector2/lane` and `/advan-ced_algorithm/polygon`. The `lane_detector_node` publishes the road lane lines spatial localisation in the image and the `advanced_algorithm_node` publishes the polygon that represents the navigable zone, which explains the name of the chosen topics

(Fig. 5.4). There are also other topics published by the processor nodes, which only exist for the purpose of viewing results such as: `/advanced_algorithm/finalResult`, `/lane_detector/result` and `/lane_detector2/result`.

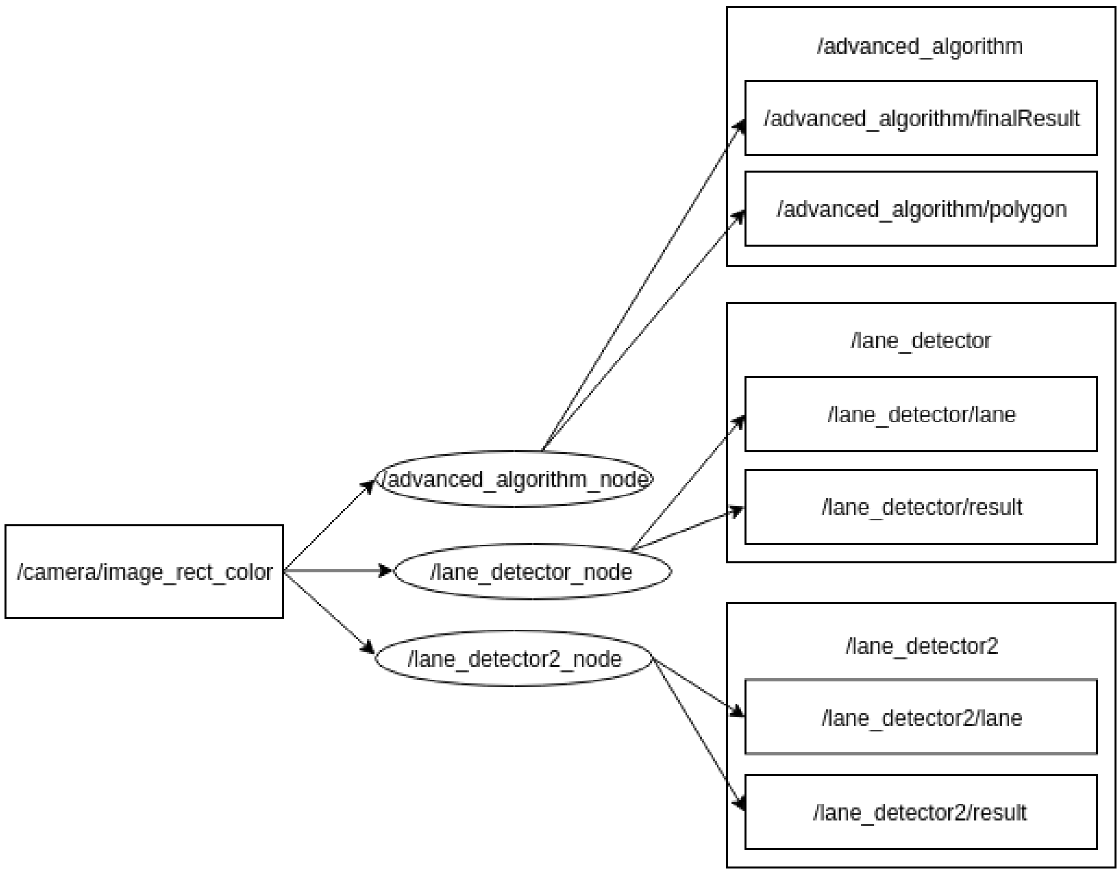


Fig. 5.4: Part of the computation graph to one camera that corresponds to the image processing for the three processor nodes.

Combining the Results of the Processing Nodes

In this phase of the architecture, the data provided by each algorithm is merged (Fig. 5.5). For this, if the previous algorithms return the spatial localisation of the lane, then there are nodes — `draw_poly_node` and `draw_poly2_node` — prepared to build the polygons from the detected lines. Thus, these nodes publish the polygons in the respective topics: `/draw_poly/poly_alg` and `/draw_poly/poly_alg2`. Finally, there is a node called `calc_prob_map_node`, which combines the polygons and publishes the result in an image (`/calc_prob_map/image_map`).

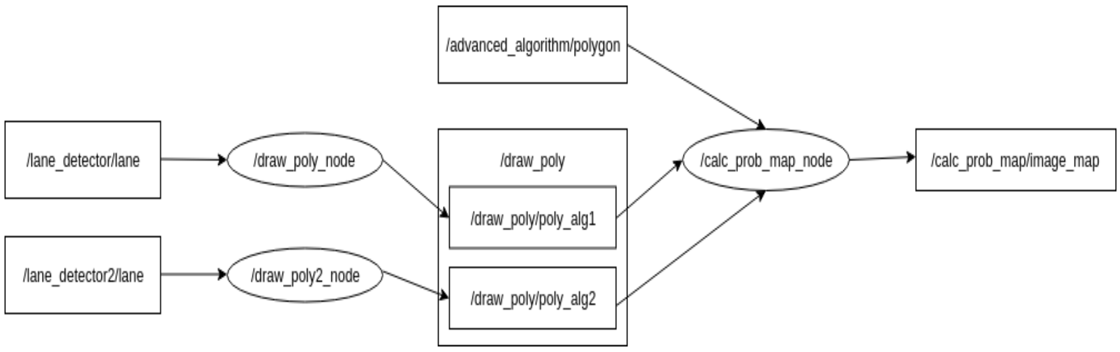


Fig. 5.5: Part of the computation graph to one camera that corresponds to the merging of information that is provided by each algorithm.

Combining the Confidence Maps provided by Multiple Cameras

The architecture so far is capable of gathering and combining information from multiple algorithms. However, a method of combining the results obtained by each camera was also developed. In short, the architecture is capable of combining results from multiple algorithms for a single camera only, as explained so far, and it is also capable of combining results (confidence maps) from each camera into a single result — multi-camera approach (Fig. 5.6). Thus, there is a node — `combine_multi_cams_node` — whose function is to combine the maps that come from multiple sources (in this work the two cameras that are used are named `top_right_camera` and `top_left_camera`). Finally, this node returns the final confidence map and publishes it in the topic `/combine_multi_cams/final_map`.

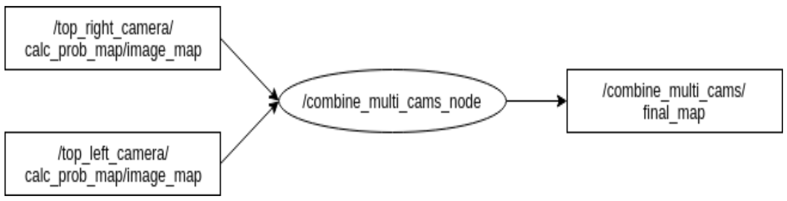


Fig. 5.6: Part of the computation graph related to the combination of the confidence maps provided by multiple cameras.

5.1.2 `Time Stamps` and Asynchronous Sources

`Time stamp` is a specific time register provided by ROS that records when an event occurred. Since this work deals with a dynamic scenario (vehicle movement), the advance in time implies an alteration of the vehicle position. Thus, the developed architecture resorts to a mechanism that controls the processing time of the frames. It enables to suppress correctly the processed frames that are lagging behind in the architecture.

Consequently, throughout the architecture, the `time stamp` of each new image is preserved. Therefore, the time of the publication of the first image is the one that is kept in the images that are created later.

In a case where the preservation of `time stamps` is not considered, the time lag between the image capture and the output of the processing algorithms — processed image — is not considered. Since the scenario where the architecture is running is dynamic, then, the architecture would return wrong data about the current road scenario. Conversely, the preservation of the `time stamps` enables to ignore processed images related to large processing time, since the current position of the vehicle is completely different from the one that was processed.

5.2 Image Processors

In this section, the implemented algorithms are explained. The first algorithm was developed by Nicolas Acero [24]. It is a ROS package, so the difficulties that were found are related to the fact that the package was prepared to ROS Indigo and this project is based on ROS Melodic. The second algorithm is based on the methodology developed by Ross Kippenbrock [32] and very similar to those developed during the course of Udacity [20]. Therefore, the algorithm was transformed into a ROS package to be part of the developed architecture. Finally, techniques based on Deep Learning are exposed as solutions to be part of the architecture in the future.

5.2.1 Algorithm 1: lane_detector ROS Package

The "lane_detector" package created and implemented by Nicolas Acero is not documented anywhere. It was found on the Github platform and it consists of several techniques developed by other authors. It starts by applying a threshold to the average of the red channel from the RGB colour space to binarise the image. After that, the IPM technique is applied based on the work developed by Mohamed Aly in [16]. The next step of the algorithm is to group the input image into horizontal and vertical lines through a smooth personalised filter. In order to validate the vertical detected lines in the previous step, a RANSAC model is performed. Finally, the lines are tracked through a combination between a Hungarian algorithm and a Kalman filter. This last step was computed by Andrey Smorodov in a multi-target tracker project [33] and adapted to this ROS package by Nicolas Acero. The lines are published through a message whose type is `geometry_msgs`. It provides messages for the coordinates of the right lane line, left lane line and guide line (a line between the lane lines).

5.2.2 Algorithm 2: advanced_lane_detection ROS Package

This algorithm was implemented to present it in a PyData conference [34]. The global PyData network promotes discussion of best practices, new approaches, and emerging technologies for data management, processing, analytics and visualisation.

The author followed a pipeline (Fig. 5.7) similar to the one presented by the authors that attended the Udacity course (mentioned in Section 2.4), which is constituted by four main steps: image rectification, image warping, lane lines segmentation and curve fitting.

Warp image application

The procedure starts with an operation to undistort the image, which is already provided by the image rectification step, after the camera calibration, as mentioned in section 4. After that, a "bird's eye view" is applied, which is a suitable technique for fitting curves and finding lane lines, because it is easier to segment them if the point of view is from

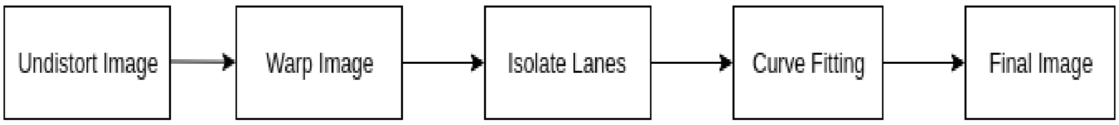


Fig. 5.7: Algorithm's pipeline that composes the "advanced_lane_detection" ROS package.

top to bottom than forward. To obtain this, there are source points in the original image, which constitute a polygon that is warped to straight lines (Fig. 5.8), through a geometric transformation between them.

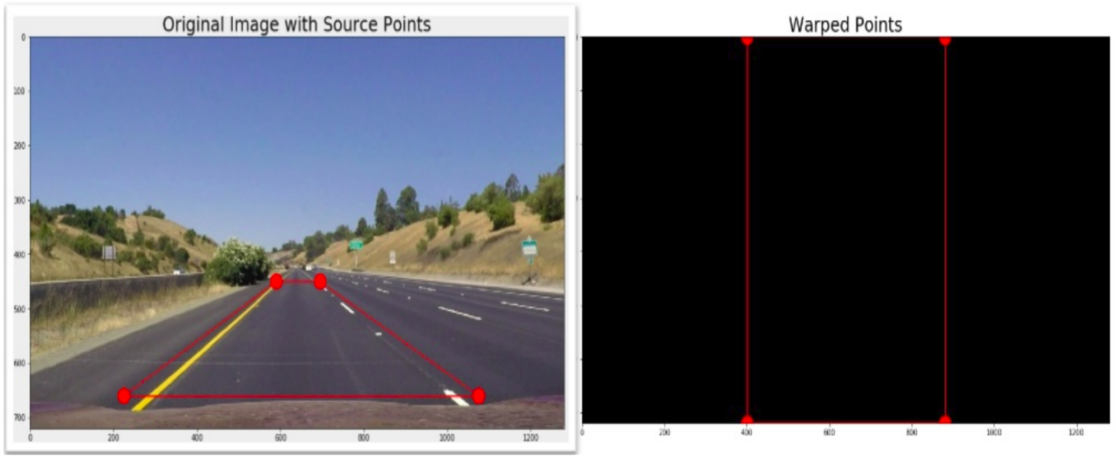


Fig. 5.8: Warping technique application [35]. The left image demonstrates the source points that are used to the warp transformation. The right image represents the warped points. Summarising, the points in the left image are transformed into the points in the right image.

Lane lines detection

Regarding the segmentation of the lane lines, there are two approaches, that were combined: "Colour Selection" and "Edge Detection". The first method — Colour Selection — consists of applying a threshold to the isolated channel — Red — from the RGB colour space.

The second implementation — Edge Detection — is constituted by the application of the Sobel Edge Detector [36]. Then, the two lane lines segmentation techniques are merged, which is extremely useful, since there is information that is provided from two different sources. This causes a sharing of information that would not be possible without the combination of the two techniques as demonstrated in Fig. 5.9.

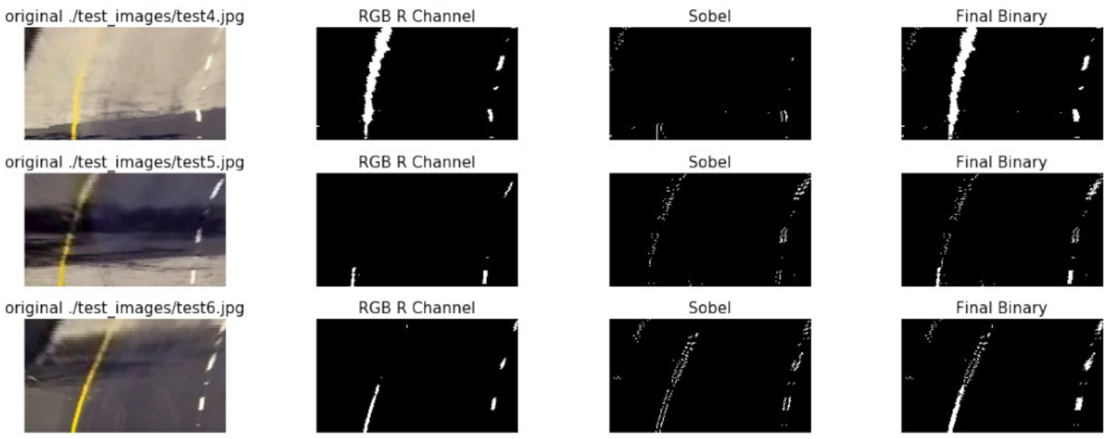

original ./test_images/test4.jpg
RGB R Channel
Sobel
Final Binary
original ./test_images/test5.jpg
RGB R Channel
Sobel
Final Binary
original ./test_images/test6.jpg
RGB R Channel
Sobel
Final Binary

Fig. 5.9: Combination of Red Channel binarisation and Sobel Edge Detector [35].

Curve fitting

Finally, a curve fitting method is applied to the previous binary image. Hence, an histogram from the bottom half of the image is taken. The histogram peaks (Fig. 5.10) represent where the lines begin, and a box is drawn in those starting points.

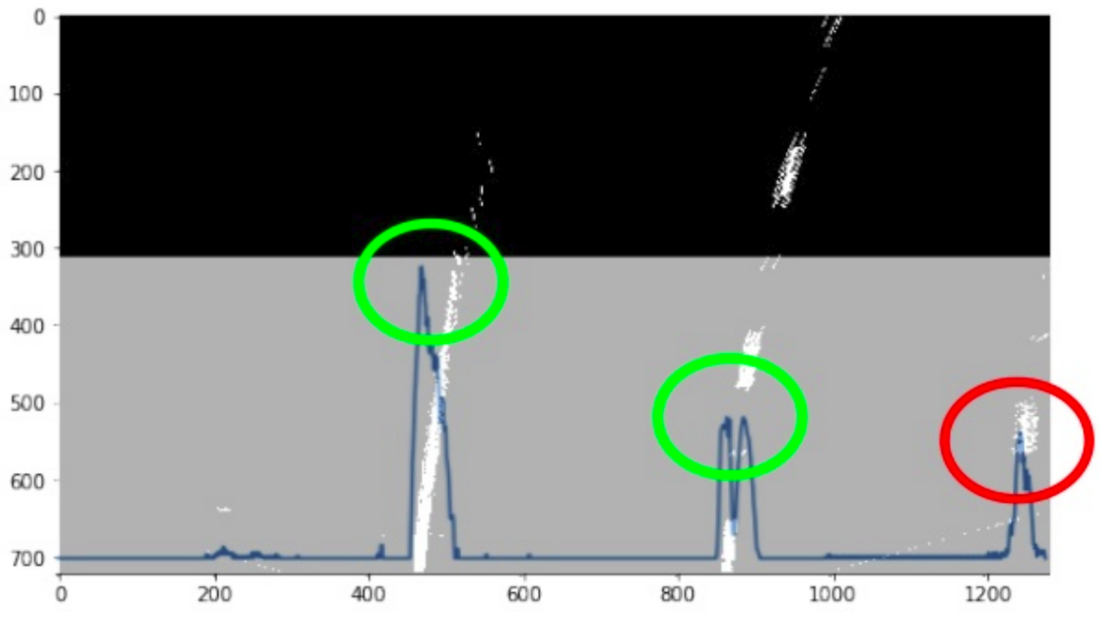


Fig. 5.10: Histogram that represents where the lane lines start [35].

Then, boxes are drawn as the line search in the image is moving up. The location of the boxes depends on the average of the previous line. Lastly, a second-order `polyfit` is applied to the pixels that were in the boxes, which gives rise to the curve fit of those points (Fig. 5.11).

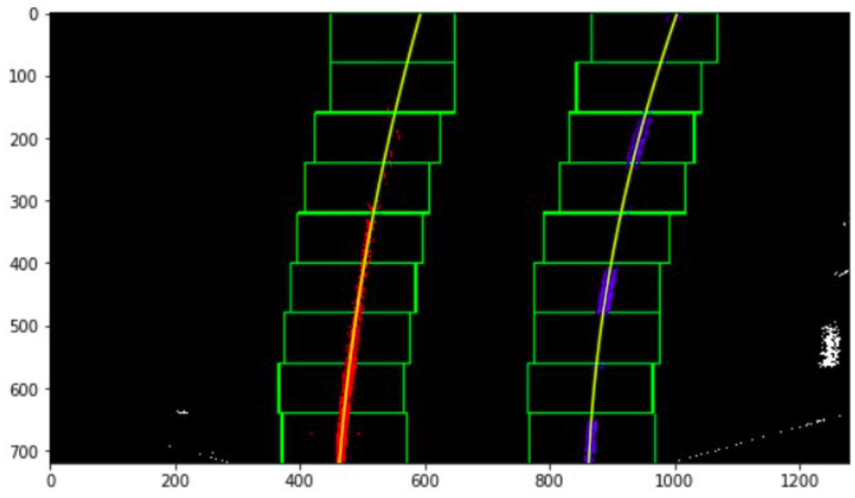


Fig. 5.11: Curve fitting procedure [35]. The green boxes are the windows that are located based on the average of the previous line.

The final image (Fig. 5.12) is obtained by applying the inverse transformation to the warped points of the first step of the procedure (”warp image application”). Therefore, a polygon is drawn between the points that were found in the previous step of the procedure, giving rise to the final representation of the algorithm.

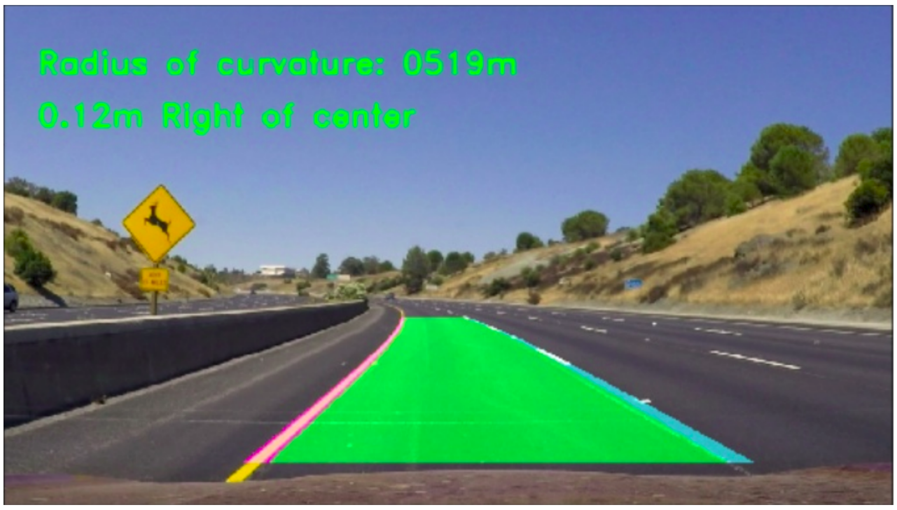


Fig. 5.12: Result obtained by applying the algorithm [35].

ROS package creation

As introduced at the beginning of this section, a ROS package was created based on this algorithm. In this regard, the code developed by Chiang Hsuche [37] was used as a library of functions to the ROS package. The developed ROS package's name is ”advanced_lane_detection” and is based on a class, which is constituted by a constructor, whose purpose is to advertise the topics that will publish the results of the algorithm. There are also some essential functions that compose the package:

- `receiveInitImg` — this function receives the rectified image and through the `CvBridge` tool converts the ROS original image message into an OpenCV image;
- `Publishers` — this function contains all the agents that will publish in topics the results of the algorithm;
- `processFrames` — this represents the core function of the package. Here the procedure of the algorithm is applied through the call of functions that are part of the library created based on the Chiang Hsuche's code.

5.2.3 Algorithms based on the Deep Learning Approach

LaneNet Ros Node

An alternative to classical techniques is an algorithm based on Deep Learning. There is a ROS package [38] based on [39]. This model (Fig. 5.13) named LaneNet consists of an encoder-decoder stage, binary semantic segmentation stage and instance semantic segmentation for a near real-time lane detection task. If the results obtained by this approach are sufficiently interesting, it would be interesting in future developments to work on the performance of the neuronal network (e.g. changing the layers) to achieve the real-time detection.

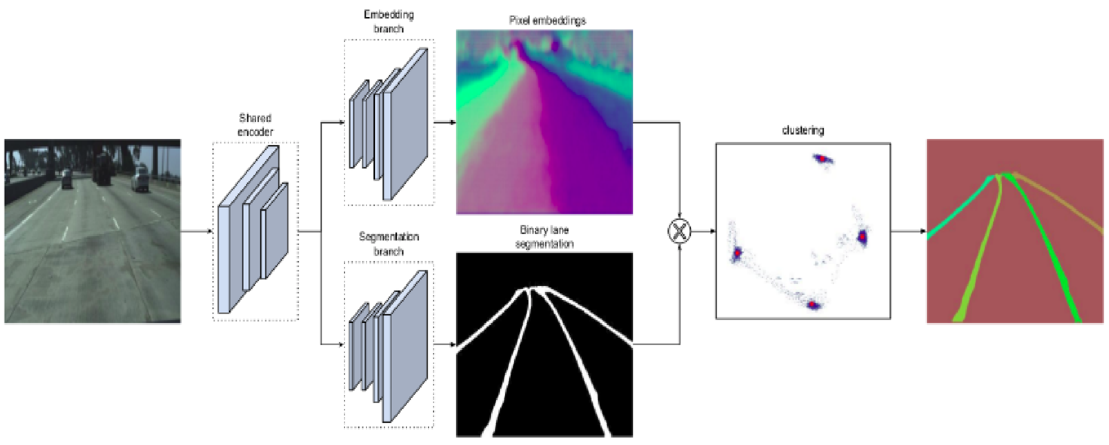


Fig. 5.13: The overview of the LaneNet architecture, as well as the visual input of each step. From the left to the right, the image enters the encoder/decoder and produces the pixel embeddings, that discriminates the lane and the lane segmentation. These are combined to produce the final result image, which is further to the right. [39].

The architecture is composed of two branches. One of them is represented at the bottom of Fig. 5.13 and its aim is to produce a binary lane mask through the network training (segmentation branch). The other branch has the goal of dividing the pixels of the lanes into groups according to the lane that they belong to (embedding branch). Through the first branch, the background pixels are suppressed from the pixel groups created in the second branch, and then, the final groups of lane lines are representing the different lane lines of the image.

Therefore, the network clusters the road lane lines: each colour of the road line result has a meaning. In a deeper view, in addition to detecting the lines of the road, the algorithm disentangles the segmented lane pixels into different lane instances (instance segmentation), through the colours that are represented in the output image (Fig. 5.14).

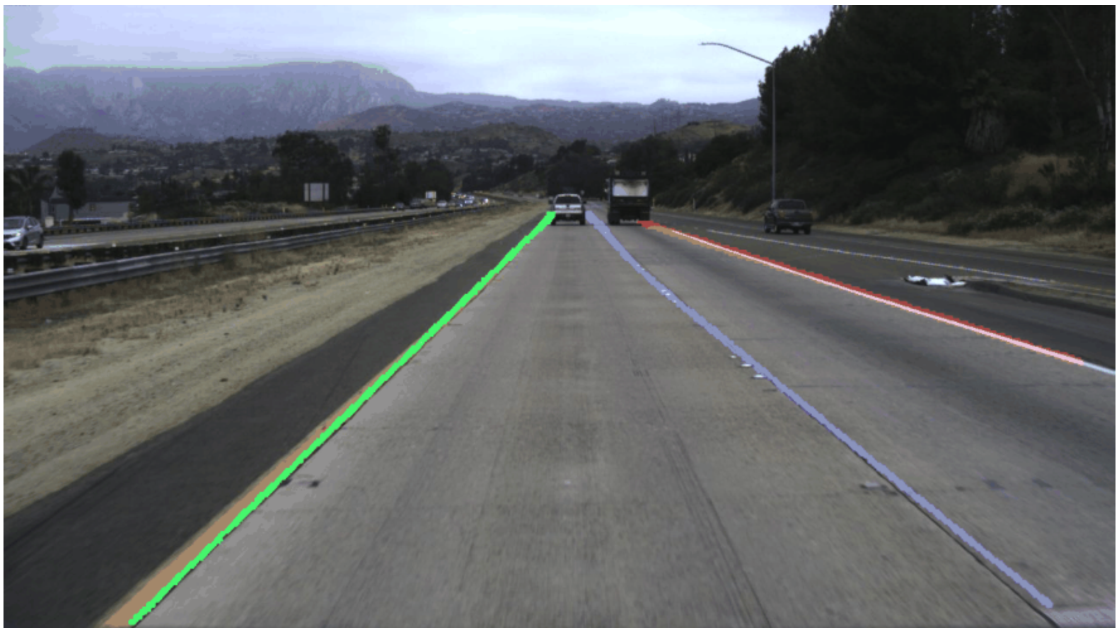


Fig. 5.14: Result obtained by applying the LaneNet ROS node approach. [38].

UNet Model

Another option in terms of the Deep Learning approach was found in the Fast AI course, which brings a set of tutorials about different techniques related to Deep Learning. The type of learning that is going to be implemented is the semantic segmentation [40] applied in a real road scenario. The multi-label feature is included in the datasets that are given before the training phase of the procedure. The other important concept related to this method is "Image Segmentation", which is the process of dividing an image into pixels groupings to then be labelled and classified. Therefore, the objects of the image that have the same colour belong to the same class.

The dataset used (Camvid dataset) is labelled according to the segmented frame (Fig. 5.15). The reason why the dataset was chosen is the Camvid is that is one of the smallest datasets available (701 labelled images divided into 32 classes), so the computational cost was low.

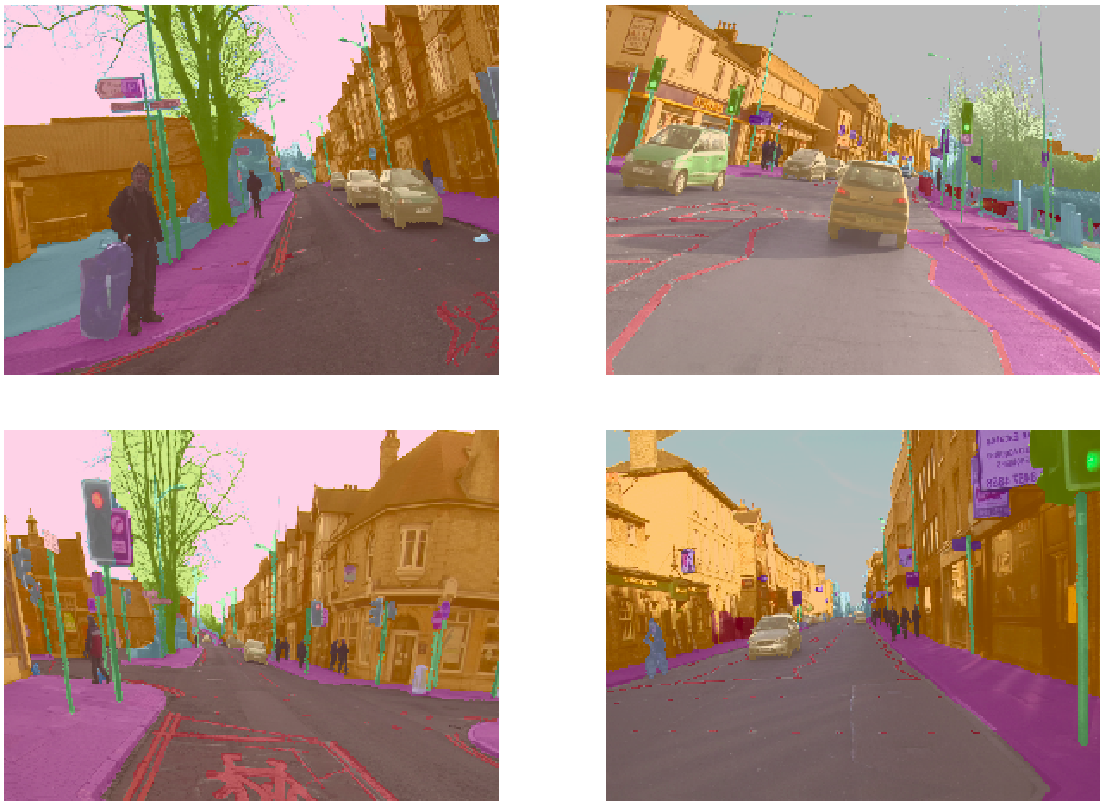


Fig. 5.15: Camvid datasets [41] used to train the architecture provided by one of the tutorials of the Fast AI course. Each colour in the image represents a different class. For example, the road is associated to the grey colour and the sidewalks are identified with the pink colour.

Also, in the experiment realised in this dissertation the classes were decreased to 11 by editing the code already provided by the Fast AI tutorials to further reduce the computational cost. There are other datasets compared in [42] that imply a higher computational cost due to their size, however, they can produce more accurate and reliable results. They are presented in the Table 5.1.

Table 5.1: Comparison of other datasets, which can be used to train the network.

Datasets	**Labelled images**	**Number of classes**
Cityscapes	3478	34
Mapillary Vistas	20000	66
ApolloScape	147000	36
BDD100K	8000	19

One interesting characteristic of the Mapillary Vistas dataset is that it includes images from cities in Portugal (Fig. 5.16), which is relevant since the ATLASCAR2 most common environment are roads in Portugal.

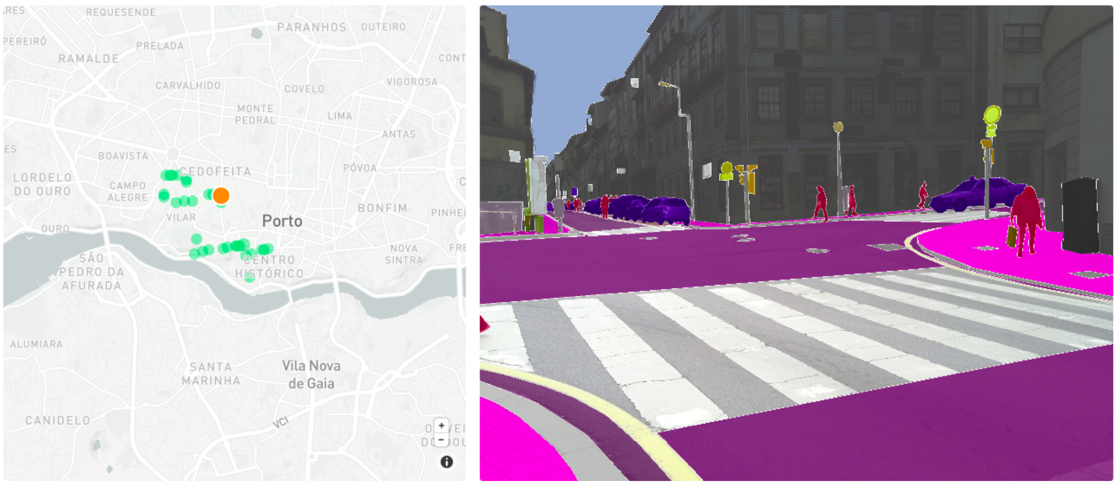


Fig. 5.16: An example of a labelled image that belongs to the Mapillary Vistas dataset. This image was captured in Porto, Portugal [43].

The training of the neural network was done in a computer present in LAR, whose main function is to work in Deep Learning thanks to the high computational power provided by their high-end GPGPU's (General Purpose Graphics Processing Unit). In order to train the network, it was necessary to work remotely and with some interesting tools like Podman [44] and Jupiter notebooks, which allows an interactive Python session in the browser [45]. Podman is a daemon-less container engine for developing, managing, and running containers. Containers, in short, contain applications in a way that keep them isolated from the host that they run on [46]. They enable applications to be packaged in *images* that contain the operating system, dependencies and configuration, so they are easy to deploy, distribute and develop. This allows easy management of resources, such as CPU (Central Process Unit) quotas, memory and visible GPU (Graphics Processing Unit) devices.

5.3 Data Combination

This section covers the core of the developed architecture — the `data_treatment` package. It launches two nodes: `draw_poly_node` and `calc_prob_map_node`. The former receives the data that comes from each algorithm, whose result is the lane lines localisation, and builds a polygon with this information. The latter creates a confidence map based on the information included on the polygons. In the multi-camera mode of the architecture, the node `"combine_multi_cams_node"` is also launched if an argument has True value on the launching of the architecture (procedure described later). This is responsible for combining the maps from the `calc_prob_map_node` result for each of the existing cameras and publishing a single confidence map.

5.3.1 Node `draw_poly_node`

This node was created in order to deal with a specific type of algorithm's result — the lines localisation in the image.

One of the implemented algorithms (the one that returns the localisation of the lines) consists of several parameters. Since these parameters are related to the visual techniques applied, the algorithm can be parameterised with different values. Thus, this algorithm originated two road lane lines detector nodes, whose results are transformed into polygons through two other nodes (`draw_poly_node`). These nodes subscribe to the `/lane_detector/lane` topic of each processor node. Since the type of messages published on these topics is associated to a ROS message (`geometry_mesgs`), they are converted into OpenCV points. It follows that the lines are drawn in an image whose size is equal to the size of the processed images. Then, the maximum and minimum points of each line are found to close the polygon. Finally, the interior of the polygon is filled through the OpenCV function "`floodfill`" (Fig. 5.17) and then published in the `/draw_poly/poly_alg1` and `/draw_poly/poly_alg2` topics. Later, these topics (polygons) will be subscribed by the node in charge of building the confidence map (`calc_prob_map_node`).

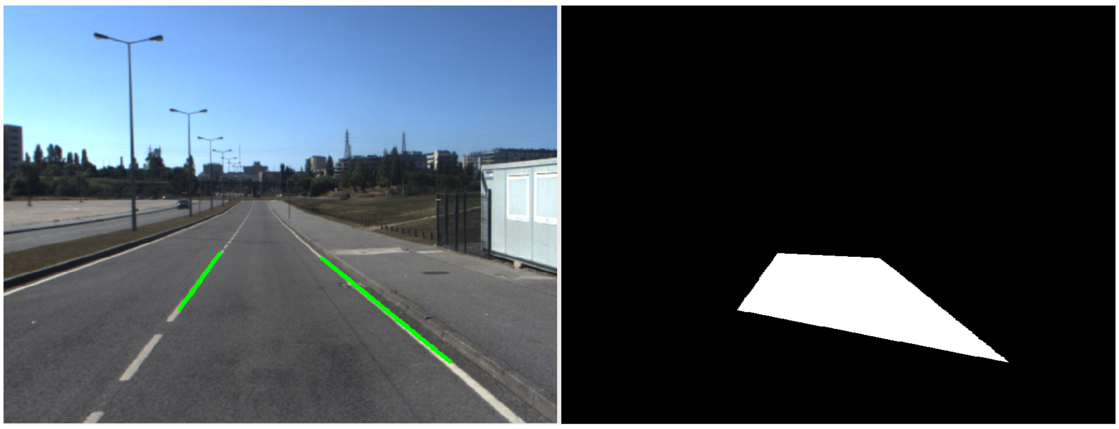


Fig. 5.17: Polygon construction from the lane lines spatial localisation.

5.3.2 Node `calc_prob_map_node`

The `calc_prob_map_node` node subscribes only to the topics that are provided in the launch file, as it is explained in the next section. It allows the architecture to receive any number of algorithms (scalability) and, consequently, enables this node to receive any number of polygons. The current state of the architecture allows launching any number of processor nodes. As a result, this node publishes the "confidence map" based on the number of topics (polygons) that it subscribes to. One case that can serve as an example is the launching of one processor node — the "`lane_detector_node`". Consequently, the launching of the architecture has to take into account some arguments, which would be provided by the following:

- with-advanced_algorithm:=false
- topics:="/top_right_camera/draw_poly/poly_alg1"

These arguments deactivate the processor node that is not used in this experiment (`advanced_algorithm_node`) and declare the topics that are going to be subscribed in order to merge the polygons built from the processor nodes. As a result, the architecture launch would be provided by the following command:

- `roslaunch data_treatment road_visual_perception.launch with-advanced-_algorithm:=false topics:="/top_right_camera/draw_poly/poly_alg1"`

The information about each polygon received by this node is stored in a structure (Listing 5.1).

Listing 5.1: Structure that contains the information about each polygon.

```
struct image_info{
    cv::Mat image;
    bool isupdate;
    double image_delay;};
```

This structure is composed of three fields: the image (in OpenCV Mat type), a Boolean variable "is_update", which represents whether the polygon was already updated or not and, finally, a Double variable "image_delay" that concerns the time it takes the whole process until the polygon reaches the part of the architecture where it is combined with other polygons. This last variable has extreme importance, because if the processing time of the original image is too long — more than 0.06 s — the polygon is not used in the following procedures. Since the camera can record 15 FPS (frames per second), it means that the processor algorithms have to be faster to process the image than the camera to capture one image. If this time interval exceeds 0.06 s, the polygon provided by the processor node is rejected because there is a possibility that the image that was processed was too outdated compared to the real scenario (as discussed in section 5.1.2). Thus, each structure represents the information of each polygon image and is stored in a vector to build the confidence map.

Regarding the construction of the confidence map, the polygons that will be used during the calculation of the confidence map are first illustrated in Fig. 5.18.

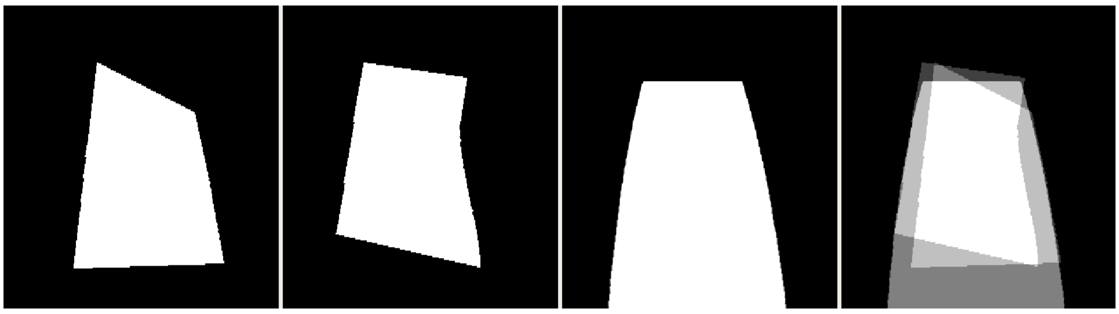


Fig. 5.18: The three polygons used to calculate the confidence map (the first three images) and a grayscale representation (fourth image), which shows the intersection (white pixels) and non-intersection (darker pixels) of those polygons. The first two images (A and B, respectively) represent the outputs of the `draw_poly_node`, which draws the polygons through the output of the `lane_detector_node` and the `lane_detector2_node`. Finally, the third image (C) illustrates the output of the `advanced_algorithm_node`.

Then, after receiving each polygon relative to the output of each algorithm, an "AND" logic operation is applied to the polygons "A", "B", and "C". This implies to obtain the intersection (D) of all polygons (Eq. 5.1).

$$D = A \wedge B \wedge C \tag{5.1}$$

This zone (D) is the one where there is more confidence that the pixels represented in white belong to the true road zone since the three algorithms have this result (Fig. 5.19).

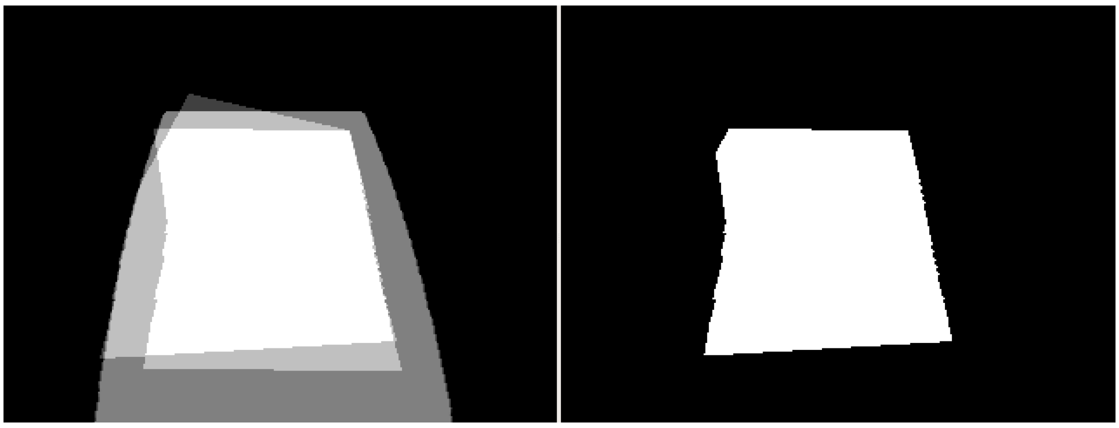


Fig. 5.19: Result obtained by applying the AND logical operation. The left image is merely illustrative of the areas where there is a larger (white pixels) and lower confidence (dark pixels) in which the pixels belong to the road lane zone.

Conversely, the non-intersection zone (E) of the three polygons is determined through the computation of the XOR logic operation (given by Eq. 5.2).

$$E = A \oplus B \oplus C \tag{5.2}$$

In this case, only one or two algorithms return this zone as a road area (white pixels in Fig. 5.20); thus, there is less confidence that this zone belongs to the driveable road zone when compared to the white pixels of the previous image.

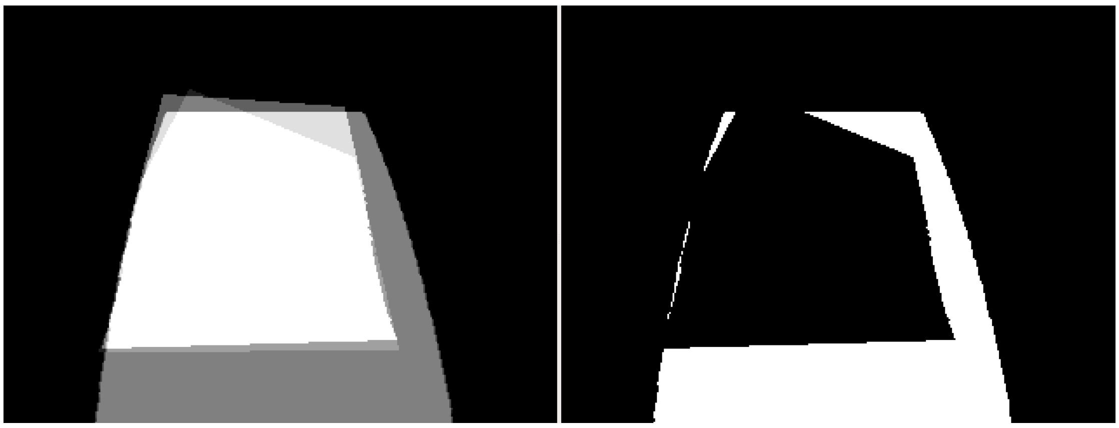


Fig. 5.20: Result obtained by applying the XOR logical operation. White pixels correspond to the result of applying the XOR operation.

Finally, the confidence map is built based on previous statements. Accordingly, a square kernel (F) is applied to the image that represents the intersection of the polygons (D). Consequently, the region where it is most possible that the pixels belong to the road area (G) is calculated. This process is given by the Eq. 5.3.

$$G = F \circledast D \tag{5.3}$$

Where F is an odd square kernel given by Eq. 5.4 in the case of filter with a size of 3×3 (this size is variable).

$$F = \begin{bmatrix} 255 & 255 & 255 \\ 255 & 255 & 255 \\ 255 & 255 & 255 \end{bmatrix} \tag{5.4}$$

As a result, each pixel value of this region is related to more or less confidence in belonging to the road zone — pixels whose values are closer to the white colour than others are more likely to be a valid representation of the road. The kernel is applied to all pixels of the image, so the confidence is given by the number of pixels of the neighbourhood that belong to the intersection area divided by the filter size. For instance, if the kernel size is given by $size(F)$, and in the neighbourhood of the pixel where the filter is applied, X pixels belong to the intersection zone. Thus, the confidence (L) that the pixel has in belonging to the correct road area is given by the Eq 5.5:

$$L = \frac{X}{size(F)} \tag{5.5}$$

However, the zone that corresponds to the non-intersection (L) of the polygons has a constant "confidence" but less than the least possible of the intersection area. Given the above, the two described zones form the confidence map (M in the Eq. 5.6), which is published as an image in the `image_map` topic.

$$M = D + L \tag{5.6}$$

As can be observed in Fig. 5.21 (`image_map` representation), the darker zones are described as zones of less confidence in belonging to the road zone and, on the contrary, the lighter areas are related to greater confidence of being a valid representation of the road driveable area.

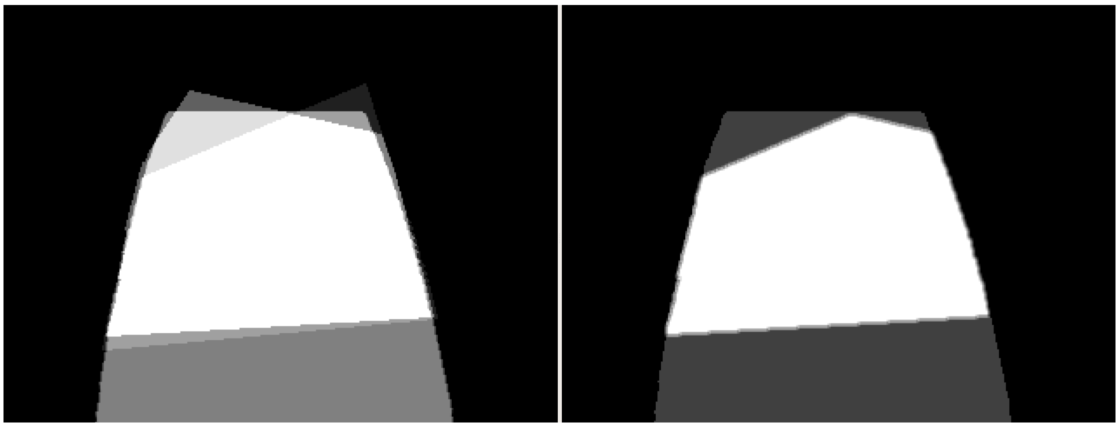


Fig. 5.21: Representation of the confidence map as an image. The darkest pixels do not belong to the intersection of the three polygons and are affected by lower confidence (dark pixels). The pixels that belong to the intersection are white (greater confidence).

There is an option implemented in this node when the architecture is running in multi-camera mode. To combine the confidence maps from multiple cameras, a perspective transformation is necessary to place the cameras into a common reference frame (`moving_axis` frame). For this, an input argument — `multi_cameras_combination` — in the architecture launching has to have True logical value. If this does not happen, the multiple cameras work independently of each other and produce multiple confidence maps viewed from different perspectives. This representation (Fig. 5.22) also corresponds to how the data from the architecture has to be published to, for example, in the future be merged into an Occupancy Grid with the data that comes from LIDAR sensors.

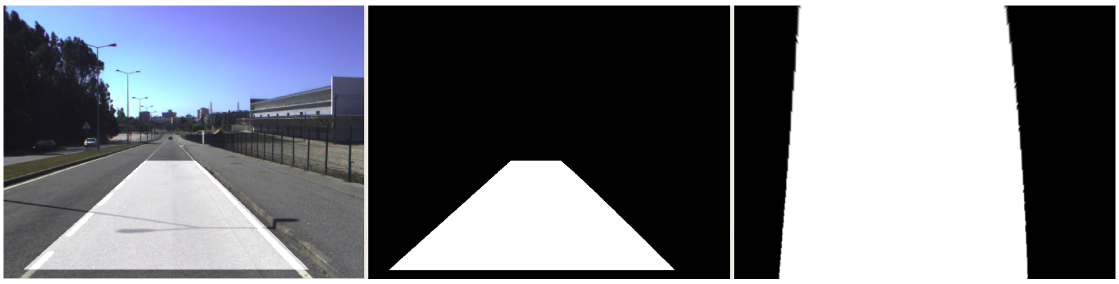


Fig. 5.22: One example of a result of a warp transformation applied to a polygon. The left image represents the algorithm detection results, which gives rise to the polygon illustrated in the centre image. Finally, the polygon warped is represented on the right image. This operation is equivalent to the one represented in Fig. 5.8.

This approach was based on the "`warpPerspective`" OpenCV function which has as input arguments a polygon that represents the road lane from the camera perspective and a perspective transformation matrix. This transformation matrix is given by the `getPerspectiveTransform` function, whose input arguments are the final and initial position of the polygon. At this stage, the representation of the final polygon (final points warped) assumes that the width of the road is 3.5 m (width of the road section which is the environment where the results of this work are analysed in the section 6). Through the intrinsic camera parameters, the units of the road lane width are transformed into pixels. This method is applied in several algorithms whose objective is to detect road lane lines (e.g. the algorithm used in this work, whose output is a polygon). However, it would be interesting to develop an IPM technique in a future work that would be more generic based exclusively on the extrinsic and intrinsic calibration of the camera.(e.g. [18]).

5.3.3 Node `combine_multi_cams_node`

The `combine_multi_cams_node` only exists when the architecture is acting in multi-camera mode. It subscribes to the nodes explained in the previous sub-section of each camera. Hence, for example, if there are two cameras (the current status of the ATLASCAR2 setup) — top_right_camera and top_left_camera — this node subscribes to the topics "`/top_right_camera/calc_prob_map/image_map`" and "`/top_left_camera/calc_prob_map/image_map`". However, the node is prepared to receive more confidence maps, it is just a matter of remapping the topics that the node has to subscribe to when the architecture is launched, as explained in the next section.

To combine the confidence maps coming from multiple cameras, these warped maps are combined through a weighted sum between the two images, where each image has the same weight.

5.4 Launching Nodes

This section details how the required nodes from the architecture are launched. There are specific files for the purpose, named *launch files*. They provide a convenient way to start up multiple nodes and a master, as well as other initialisation requirements such as setting parameters [47]. Finally, in the last sub-section, the use of two cameras will be summarised.

5.4.1 Main Launch Files

In order to divide the launching of the architecture, there are two main launch files: `drivers.launch` and `road_visual_perception.launch`. As the name indicates, the `drivers.launch` file launches all the ATLASCAR2 sensors, which include the sensors that are part of the architecture — the two cameras.

The `road_visual_perception.launch` (Listing 5.2) launches the nodes corresponding to each algorithm (`algorithm1.launch and advanced_lane_detection.launch`), the nodes that draw the polygons through the information that is obtained from the algorithms that return the road lines localisation in the image (`draw_poly.launch`) and the node that joins the information from each processor node (`data_treatment.launch`).

Listing 5.2: `road_visual_perception.launch` file that launches all the architecture.

```
<launch>
    <arg name="with−advanced_algorithm" default="true"/>
    <arg name="with−nsteel_algorithm" default="true"/>
```

```xml
    <arg name="topics_left" default="/top_left_camera/draw_poly/poly_alg1,/top_left_camera/draw_poly/poly_alg2,/top_left_camera/advanced_algorithm/polygon"/>
    <arg name="topics_right" default="/top_right_camera/draw_poly/poly_alg1,/top_right_camera/draw_poly/poly_alg2,/top_right_camera/advanced_algorithm/polygon"/>
    <arg name="top_left_camera" default="true"/>
    <arg name="top_right_camera" default="true"/>
    <arg name="multi_cameras_combination" default="true"/>
    <arg name="topics_maps" default="/top_left_camera/calc_prob_map/image_map,/top_right_camera/calc_prob_map/image_map"/>

    <!-- To launch the nsteel algorithm node -->
    <include file="$(find lane_detector)/launch/algorithm1.launch" if="$(arg with-nsteel_algorithm)">
        <arg name="top_left_camera" value="$(arg top_left_camera)"/>
        <arg name="top_right_camera" value="$(arg top_right_camera)"/>
    </include>

    <!-- To launch the advanced algorithm node  -->
    <include file="$(find advanced_lane_detection)/launch/advanced_lane_detection.launch" if="$(arg with-advanced_algorithm)">
        <arg name="top_left_camera" value="$(arg top_left_camera)" />
        <arg name="top_right_camera" value="$(arg top_right_camera)"/>
    </include>

    <!-- Just to draw the polygons to the algorithms that return localisation lanes -->
    <include file="$(find data_treatment)/launch/draw_poly.launch" if="$(arg with-nsteel_algorithm)">
        <arg name="top_left_camera" value="$(arg top_left_camera)"/>
        <arg name="top_right_camera" value="$(arg top_right_camera)"/>
    </include>

    <!-- To launch the nodes that calculates the confidence maps -->
    <include file="$(find data_treatment)/launch/data_treatment.launch">
        <arg name="topics_left" value="$(arg topics_left)" if="$(arg top_left_camera)"/>
        <arg name="topics_right" value="$(arg topics_right)" if="$(arg top_right_camera)"/>
        <arg name="top_left_camera" value="$(arg top_left_camera)"/>
        <arg name="top_right_camera" value="$(arg top_right_camera)"/>
        <arg name="multi_cameras_combination" value="$(arg multi_cameras_combination)"/>
        <arg name="topics_maps" value="$(arg topics_maps)" if="$(arg multi_cameras_combination)"/>
    </include>
</launch>
```

This launch file enables to launch one or two processor algorithms, due to the Boolean arguments (`with-advanced_algorithm` and `with-nsteel_algorithm`). These have their logic values depending on the use of them in the processing stage. In addition to this, there are also two arguments (`topics_right` and `topics_left`) associated to the `data_treatment.launch` file, whose function is to declare the topics (related to each camera) that are subscribed by the node responsible for the confidence map construction. Besides that, this launch file takes into account which camera is being used (`top_left_camera` and `top_right_camera` arguments) and in case they are both used, whether the combination of maps coming from each camera is made or not (`multi_cameras_combination` argument). Finally, there is an argument (`topics_maps`) that similarly to what happens with the topics (`topics_right` and `topics_left`) that are subscribed by the node that calculates the confidence map for each camera, translates the topics inherent to the confidence maps that will be subscribed by the node that combines the warped maps published in the case of multiple cameras.

Therefore, to launch the sensors and the architecture, it is necessary to run the following commands:

1. `roslaunch atlas2_bringup drivers.launch`;

2. `roslaunch data_treatment road_visual_perception.launch` — may have optional arguments related to the image that is used in terms of source (if it is used the images captured by the two cameras or not), the algorithms that will process the image, the topics inherent to the polygons that are combined, whether exists combination of multiple cameras or not and the topics related to the combination of the confidence maps (multi-camera mode).

Where `data_treatment` is the package in charge of merging the information coming from each algorithm and for the combination of the confidence maps provided by each camera (multi-camera mode). The package name is followed by the main launch file of the architecture and then by the existing input arguments.

5.4.2 Architecture Launch Files

As mentioned before, there are three files launched from the main architecture file — `road_visual_perception.launch`. The first one (Listing 5.3) is related to the first implemented algorithm developed by Nicolas Acero. The package promotes the possibility of processing images stored on disk, instead of analysing frames from cameras; it was modified with "`remaps`" to provide the correct name to the topics. Also, it takes into account parameters that provide the resize of the initial image to get a faster processing time, as explained in section 5.3. Finally, the processor node is cloned by creating the node "`lane_detector2`", which loads different parameters related to the algorithm performance (e.g. points of the ROI (Region of Interest) used in the IPM technique).

Listing 5.3: This is an example of the `algorithm1.launch` that launches the two processor nodes whose output are the localisation of the road lane lines in the image (here only one processor node is launched in order to be easier to understand).

```
<launch>
  <arg name="load_images_from_folder" default="false" />
  <arg name="top_left_camera" default="true"/>
  <arg name="top_right_camera" default="true"/>

  <node pkg="lane_detector" type="lane_detector" name="lane_detector_node"
    output="screen" ns="top_left_camera" if="$(arg top_left_camera)">
    <remap from="camera_info" to="/top_left_camera/camera_info" />
    <remap from="/image" to="/top_left_camera/image_rect_color" />
    <param name="images_from_folder" value="$(arg load_images_from_folder)" /
    >
    <param name="images_path" value="$(find lane_detector)/data/first_set" />
    <remap from="lane_detector/processed" to="/top_left_camera/lane_detector/processed" />
    <remap from="lane_detector/lane" to="/top_left_camera/lane_detector/lane" />
    <remap from="lane_detector/result" to="/top_left_camera/lane_detector/result"/>
    <param name="~cols_resize" type= "int" value="964"/>
    <param name="~rows_resize" type= "int" value="720"/>
  </node>

  <node pkg="lane_detector" type="lane_detector" name="lane_detector_node" output="screen" ns="top_right_camera" if="$(arg top_right_camera)">
    <remap from="camera_info" to="/top_right_camera/camera_info" />
    <remap from="/image" to="/top_right_camera/image_rect_color" />
    <param name="images_from_folder" value="$(arg load_images_from_folder)" />
    <param name="images_path" value="$(find lane_detector)/data/first_set" />
    <remap from="lane_detector/processed" to="/top_right_camera/lane_detector/processed" />
    <remap from="lane_detector/lane" to="/top_right_camera/lane_detector/lane" />
    <remap from="lane_detector/result" to="/top_right_camera/lane_detector/result"/>
    <param name="~cols_resize" type= "int" value="964"/>
    <param name="~rows_resize" type= "int" value="720"/>
  </node>
</launch>
```

The next launch file (Listing 5.4) — `advanced_lane_detection.launch` — is associated to the ROS package that was created to perform as a second algorithm. It is a simple sample of a launch file because it only includes the node launching and two parameters, which are related to the resizing of the original image:

- `cols_resize` — represents the number of columns of the image that is processed by each algorithm;
- `rows_resize` — represents the number of rows of the image that is processed by each algorithm.

Listing 5.4: `advanced_lane_detection.launch` file whose function is to launch the `advanced_algorithm_node`.

```
<launch>
    <arg name="top_left_camera" default="true"/>
    <arg name="top_right_camera" default="true"/>

    <!-- Node to top_left_camera -->
    <node pkg="advanced_lane_detection" type="main" name="advanced_algorithm_node" output="screen" ns="top_left_camera" if="$(arg top_left_camera)">
        <remap from="/camera/image_rect_color" to="/top_left_camera/image_rect_color" />
        <remap from="/advanced_algorithm/polygon" to="/top_left_camera/advanced_algorithm/polygon"/>
        <remap from="/advanced_algorithm/finalResult" to="/top_left_camera/advanced_algorithm/finalResult"/>
        <param name="~cols_resize" type= "int" value="964"/>
        <param name="~rows_resize" type= "int" value="720"/>
    </node>

    <!-- Node to top_righ_camera -->
    <node pkg="advanced_lane_detection" type="main" name="
    advanced_algorithm_node" output="screen" ns="top_right_camera" if="$(arg
    top_right_camera)">
        <remap from="/camera/image_rect_color" to="/top_right_camera/
    image_rect_color" />
        <remap from="/advanced_algorithm/polygon" to="/top_right_camera/
    advanced_algorithm/polygon"/>
        <remap from="/advanced_algorithm/finalResult" to="/top_right_camera/
    advanced_algorithm/finalResult"/>
        <param name="~cols_resize" type= "int" value="964"/>
        <param name="~rows_resize" type= "int" value="720"/>
    </node>
</launch>
```

If the processor nodes return the lane line localisation in the image, then some nodes draw the polygon based on that. To this, the file presented on the Listing 5.5 launches the nodes responsible for this. This *launch file*, analogously to the previous one, includes some parameters related to the resize of the image and the respective **`remaps`**.

Listing 5.5: **`draw_poly.launch`** file (example of the launch file for the launch of just one node "**`draw_poly_node`**" that will build the polygons based on the **`lane_detector_node`** outputs).

```
<launch>
    <arg name="top_left_camera" default="true"/>
    <arg name="top_right_camera" default="true"/>
    <!-- Draw poly node to the first processor node that return lnes_:
    top_left_camera -->
    <node pkg="data_treatment" name="draw_poly_node" type="subscriber" output
    ="screen" ns="top_left_camera" if="$(arg top_left_camera)">
        <remap from="lane_detector/lane" to="/top_left_camera/lane_detector/
    lane"/>
        <remap from="draw_poly/poly_alg1" to="/top_left_camera/draw_poly/
    poly_alg1"/>
        <remap from="camera/image_raw" to="/top_left_camera/image_raw"/>
        <param name="~cols_resize" type= "int" value="964"/>
        <param name="~rows_resize" type= "int" value="720"/>
    </node>

    <!-- Draw poly node to the first processor node that return lnes_:
    top_right_camera -->
    <node pkg="data_treatment" name="draw_poly_node" type="subscriber" output
    ="screen" ns="top_right_camera" if="$(arg top_right_camera)">
        <remap from="lane_detector/lane" to="/top_right_camera/lane_detector/
    lane" />
        <remap from="draw_poly/poly_alg1" to="/top_right_camera/draw_poly/
    poly_alg1"/>
        <remap from="camera/image_raw" to="/top_right_camera/image_raw"/>
        <param name="~cols_resize" type= "int" value="964"/>
        <param name="~rows_resize" type= "int" value="720"/>
    </node>
</launch>
```

Regarding the core of the architecture, there is a file (Listing 5.6) that launches the main node `calc_prob_map_node`, which, similarly to the ones presented before, incorporates two image parameters related to the original image re-scale (`cols_img_small` and `rows_image_small`). It also has more three different parameters:

- `topics_polygons` — includes a `String` of the topics that are subscribed by the `calc_prob_map_node`. It allows to run any number of algorithms, that process the original image;
- `cols_img_big` — represents the number of columns of the image that is captured by the camera (chosen in launch file of the used sensors), whose function is to get the scale factor of the original image reduction;
- `kernel_size` — is the size of the kernel used to filter the intersection zone of the polygons to build the confidence map of the lane.

Moreover, this file launches the node that combines the confidence maps that come from multiple cameras (`combine_multi_cams_node`). This node has an important parameter — `topics_maps` — which represents the topics that will be subscribed by the node responsible for combining the confidence maps. Finally, this file also launches a node named "`dynamicParameters`", which through the *Dynamic Reconfigure* ROS module enables to change the `kernel_size` parameter in real-time.

Listing 5.6: `data_treatment.launch` file that launches all the nodes which are involved in building the final confidence map.

```
<launch>
    <arg name="top_left_camera" default="true"/>
    <arg name="top_right_camera" default="true"/>
    <arg name="multi_cameras_combination" doc="enables the combination of two
     confidence maps from multi-cameras"/>


    <!-- Node that calculate and launches the confidence map -->
    <!-- Top left camera -->
    <arg name = "topics_left" doc="topics that that represents the polygons
    to the left camera"/>
    <node pkg="data_treatment" name="calc_prob_map_node" type="junction_data"
     output="screen" ns="top_left_camera" if="$(arg top_left_camera)">
        <remap from="calc_prob_map/image_map" to="/top_left_camera/
    calc_prob_map/image_map"/>
        <remap from="/camera/camera_info" to="/top_left_camera/camera_info" /
    >
        <param name="~topics_polygons" type="string" value="$(arg topics_left
    )"/>
        <param name="~cols_img_big" type= "double" value="964"/>
        <param name="~cols_img_small" type= "double" value="964"/>
        <param name="~rows_img_small" type= "double" value="720"/>
    </node>

    <!-- Top right camera -->
    <arg name = "topics_right" doc="topics that represents the polygons to
    the right camera"/>
    <node pkg="data_treatment" name="calc_prob_map_node" type="junction_data"
     output="screen" ns="top_right_camera" if="$(arg top_right_camera)">
```

```
        <remap from="calc_prob_map/image_map" to="/top_right_camera/calc_prob_map/image_map"/>
        <remap from="/camera/camera_info" to="/top_right_camera/camera_info"/>
        <param name="~topics_polygons" type="string" value="$(arg topics_right)"/>
        <param name="~cols_img_big" type= "double" value="964"/>
        <param name="~cols_img_small" type= "double" value="964"/>
        <param name="~rows_img_small" type= "double" value="720"/>
    </node>

    <!-- Launching the combination of two cameras node-->
    <arg name = "topics_maps" doc="topics that represents the maps of each camera"/>
    <node pkg="data_treatment" name="combine_multi_cams_node" type="two_cameras_combination" output="screen" if="$(arg multi_cameras_combination)">
        <param name="~topics_maps" type="string" value="$(arg topics_maps)"/>
        <param name="~cols_img_small" type= "double" value="964"/>
        <param name="~rows_img_small" type= "double" value="720"/>
    </node>

    <node pkg="data_treatment" name="dynamicParameters" type="dynamicParameters" output="screen"/>
</launch>
```

5.4.3 Launch Files Summary

To summarise, the architecture is prepared to launch one camera and one or two processor algorithms as well as two cameras and one or two processor algorithms through argument remaps in the architecture launching. For example, in the case of the launching of the `advanced_algorithm_node` and only the image captured by the `top_left_camera` is used in the processing stage, then the launching node would be provided by the following command:

- `roslaunch data_treatment road_visual_perception.launch top_right_camera:=false with-nsteel_algorithm:=false topics_left:="/top_left_camera/advanced_algorithm/polygon" multi_cameras_combination:=false`

The default values of the arguments of the architecture give rise to the use of two cameras and all algorithms (that is why, in the previous example, the argument `multi_cameras_combination` is false). In order to clarify the possibilities of launching nodes of the developed architecture, Table 5.2 is presented.

Table 5.2: Architecture arguments and corresponding values

Arguments	**Description**	**Value**
with-advanced_algorithm	If the processor node relative to the advanced_algorithm is launched.	true/false
with-nsteel_algorithm	If the processor node relative to the Nsteel's algorithm is launched.	true/false
topics_left	Topics that are going to be subscribed by the node that builds the map (left camera as source).	String topics_left
topics_right	Topics that are going to be subscribed by the node that builds the map (right camera as source).	String topics_right
top_left_camera	Whether the image captured by the left camera is used or not.	true/false
top_right_camera	Whether the image captured by the right camera is used or not.	true/false
multi_cameras_combination	Whether the combination of the maps that come from different sources is made or not.	true/false
topics_maps	Topics that are going to be subscribed by the node that builds the final combined map.	String topics_map

Where "String topics_left" (it implies that the `top_left_camera` argument is equal to true) can be given by:

- `"/top_left_camera/draw_poly/poly_alg1,/top_left_camera/draw_poly/poly_alg2,/top_left_camera/advanced_algorithm/polygon"` (default value);
- `"/top_left_camera/draw_poly/poly_alg1,/top_left_camera/draw_poly/poly_alg2"`, if only the `with-nsteel_algorithm` argument is true;
- `"/top_left_camera/advanced_algorithm/polygon"`, if only the `with-advanced_algorithm` variable is true;

And "String topics_right" (it implies that the `top_right_camera` argument is equal to true) can be given by:

- `"/top_right_camera/draw_poly/poly_alg1,/top_right_camera/draw_poly/poly_alg2,/top_right_camera/advanced_algorithm/polygon"` (default value);
- `"/top_right_camera/draw_poly/poly_alg1,/top_right_camera/draw_poly/poly_alg2"`, if only the `with-nsteel_algorithm` argument is true;

- `"/top_right_camera/advanced_algorithm/polygon"`, if only the `with-advanced-_algorithm` variable is true;

And "String topics_maps" (it implies that the arguments: `multi_cameras_combination`, `top_right_camera` and `top_left_camera` are equal to true) can be given by:

- `"/top_left_camera/calc_prob_map/image_map,/top_right_camera/calc_prob_-map/image_map"` (default value and the only one possible since there are only two cameras prepared in the ATLASCAR2 setup);

Intentionally blank page.

Chapter 6

Experiments and Results

This chapter presents the experiments made to prove the scalability and reliability of the developed architecture. Therefore, the results obtained using one and two cameras combined with the use of one or two algorithms are shown.

First, the algorithms performance are discussed, as well as the difficulties/problems found in terms of implementation and parametrisation. This discussion/description is only present in the section related to the conjugation of one camera with each of the algorithms (section 6.1) and in the section that describes the results obtained by applying the Deep Learning techniques (section 6.5).

Apart from the performance of the algorithm, the usefulness, scalability and reliability of the proposed architecture are demonstrated by carrying out several experiments. They range from the simple "one camera–one algorithm" up to "multiple cameras–multiple algorithms". The experiments do not assess the architecture directly, but show how the architecture can be used to test the performance of the algorithms and their combination. Therefore, performance indicators were created to allow the evaluation and tuning of the algorithms depending on some variable parameter.

The proposed metrics are based on the areas of the confidence maps described in section 5.3.2, and the variable parameter is the size of the smoothing filter used (variation from 3×3 to 51×51). The first indicator (I_1) is defined by the Eq. 6.1:

$$I_1 = \frac{WCA}{A_T} \tag{6.1}$$

Where:

- WCA is the "weighted confidence area" of the confidence map. Mathematically, it corresponds to the sum of the value of all pixels that belong to the confidence map (M). M is the normalised matrix of the confidence map image (this means that the range values of the pixels are from 0 to 1). Therefore, WCA is calculated through the Eq. 6.2, where n_c and n_r are the positions along the columns and rows of the image matrix of each pixel, respectively.

$$WCA = \sum_{n_r} \sum_{n_c} (M(n_r, n_c)) \tag{6.2}$$

- A_T is the total area of the confidence map (Eq. 6.3) and the variables n_c and n_r have the same meaning as the one presented in the Eq. 6.2.

$$A_T = \sum_{n_r} \sum_{n_c} \lceil M(n_r, n_c) \rceil = \sum_{n_r} \sum_{n_c} \text{ceil}\,(M(r, c)) \tag{6.3}$$

The second performance index (I_2) is given by the Eq. 6.4:

$$I_2 = \frac{A_C}{A_T} \tag{6.4}$$

Where:

- A_C is the common area. It is related to the area of the confidence map where all pixels have a 100 % confidence in belonging to the driveable zone (white pixels). Consequently, A_C is given by the Eq. 6.5 (the variables n_c and n_r have the same meaning as the one presented in the Eq. 6.2):

$$A_C = \sum_{n_r} \sum_{n_c} \lfloor M(n_r, n_c) \rfloor = \sum_{n_r} \sum_{n_c} \text{floor}\,(M(n_r, n_c)) \tag{6.5}$$

- A_T is the total area of the confidence map (in pixels) — as demonstrated in Eq. 6.3.

All experiments evaluated in terms of the architecture results, were analysed in the same road section (Fig. 6.1), to the comparison between each case to be reliable.

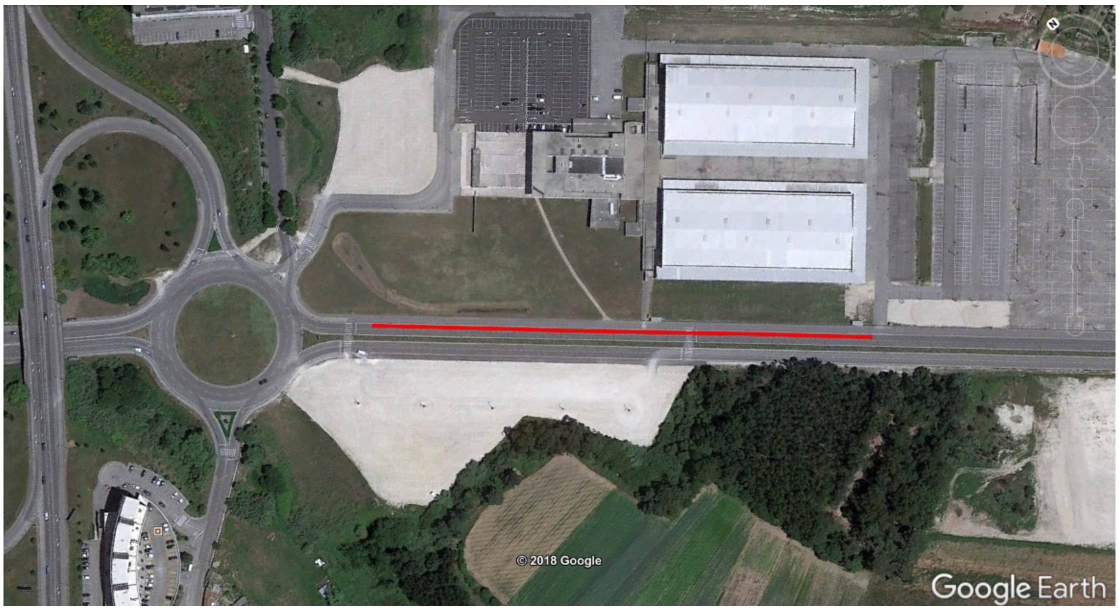


Fig. 6.1: The road section is marked in red in this image. It was the environment where the experiments presented in the following sections were made. This road section has generated 360 frames to be processed.

In summary, there will be two types of discussions in this chapter: the first is related to the quality of the algorithms implemented and the second is related to the combination of algorithms through the developed architecture.

6.1 One Camera and One Algorithm

This experiment concerns the usage of a single camera with one processor algorithm. The results obtained by the processor `lane_detector_node` are presented, following the conclusions about the `advanced_algorithm_node`. The first part of each sub-section consists of a discussion about the general behaviour of the algorithm, and the next part is related to the results of the architecture. Here, the results obtained for each of the indicators are shown.

6.1.1 Active Processor Node: `lane_detector_node`

Algorithm Behaviour

The first experiment consists of the usage of the processor node "`lane_detector_node`". Regarding the practical results of the algorithm, it demonstrated interesting results for the section of road assessed (Fig. 6.2).

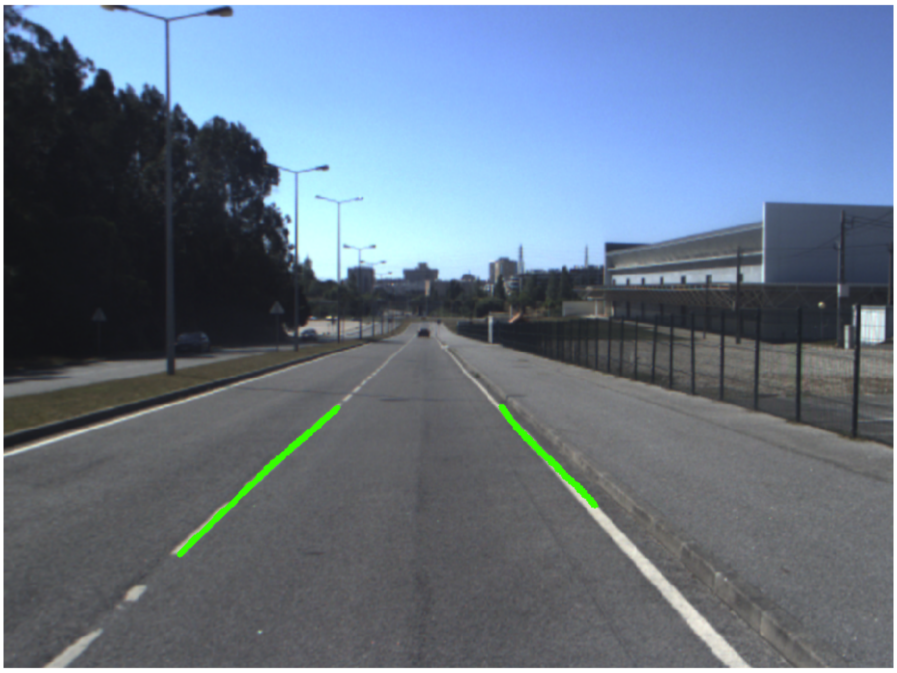


Fig. 6.2: One of the results obtained by using the `lane_detector_node`.

However, it is an algorithm that uses a large number of parameters, which made it difficult to parameterise. Moreover, it is important to characterise it as not very generic in terms of road types, since it takes as parameters the geometry of the road lane. Nevertheless, at the level of road lines detection, after the algorithm has been parameterised, it demonstrated consistent results. Even with the appearance of roadblocks and walkways, the algorithm did not show negative results (Figs. 6.3 and 6.4).

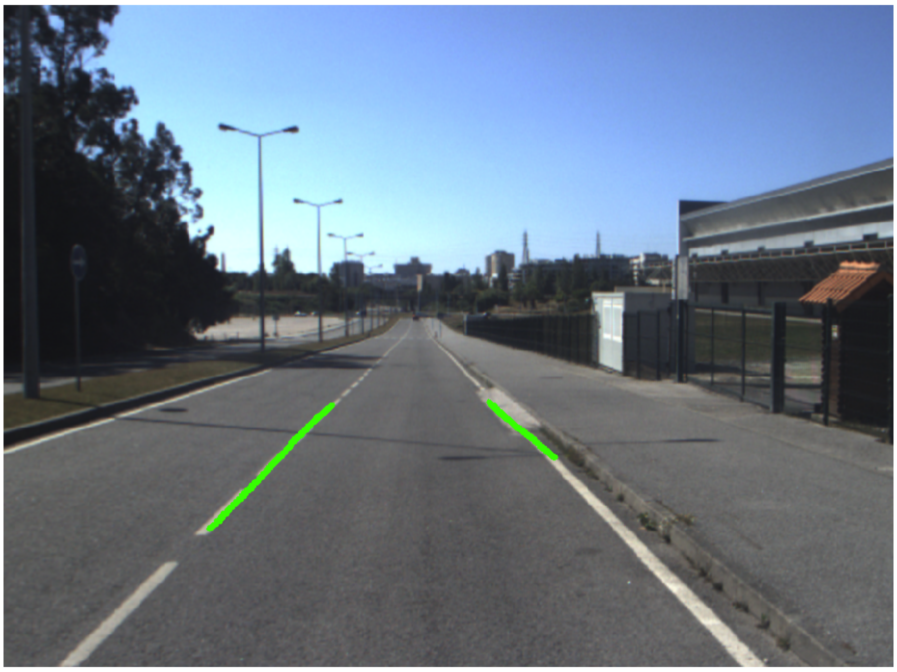


Fig. 6.3: This image illustrates the case of a right-hand side occlusion. This occlusion was not a limitation for the algorithm to detect the road lane lines.

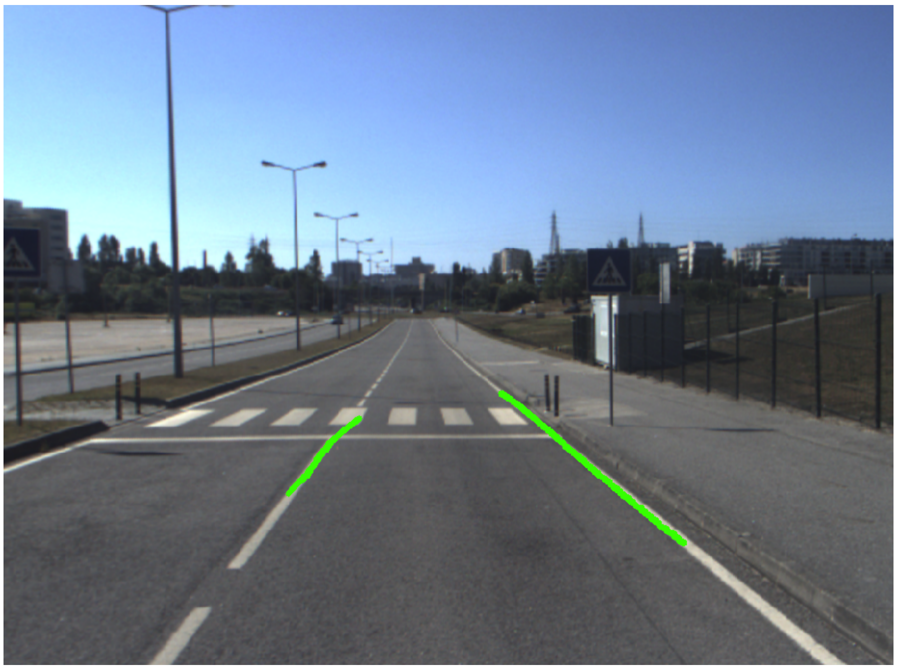


Fig. 6.4: This image represents the case of a crosswalk, which was also a situation that the algorithm could handle well.

One of the negative points demonstrated by this algorithm and that affected the processing time of the architecture was the non-detection of the road lines when the image sizes were less than 964 × 720. This could possibly be due to a decrease in the width of the road line. Another repercussion of the large processing time is that it makes the algorithm's result useless as it takes longer to process than the camera's image capture time. Consequently, the condition of maximum processing time (mentioned in section 5.3.2) was removed when this algorithm was analysed.

Architecture Results

This experiment consists of using the `lane_detector_node` as the processor of the images captured from a single camera.

Two examples of the road lines detection and the respective confidence map are shown in Fig. 6.5. Each example corresponds to the use of a filter with the minimum and maximum size of the evaluated value range.

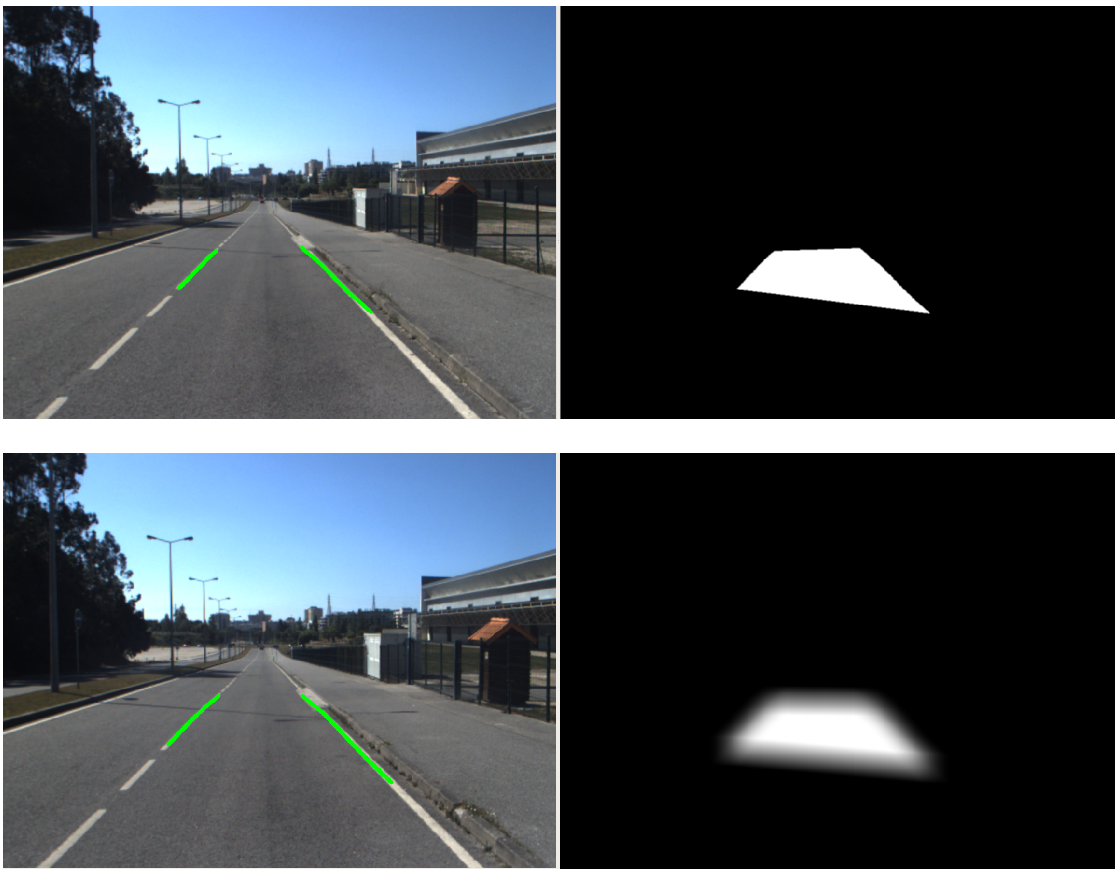


Fig. 6.5: The image in the upper right corner represents the use of a filter with a size of 3 × 3 for the construction of the confidence map and the image in the right lower corner illustrates the use of a filter of 51 × 51. The left images represent the road lane lines detection for each case. The difference between the filter sizes is found in the blur around the area represented with white pixels on the confidence maps.

The numeric results are presented in graphs. The first result presented is related to the variation of the areas (Fig. 6.6) used for the calculation of each indicator. Therefore, the following parameters are calculated for all the frames that compose the section of road evaluated:

- WCA - the "weighted confidence area" of each frame is calculated and at the end of each road section the average of this parameter in all frames — WCA_{avg} — is calculated;
- A_c - the common area of the confidence map is also calculated to each frame. Analogously to the WCA parameter, the average of each road section —$A_{c,avg}$ —is used to compare each obtained result;
- A_t - the total area of the confidence map is treated similarly to the parameters mentioned above.

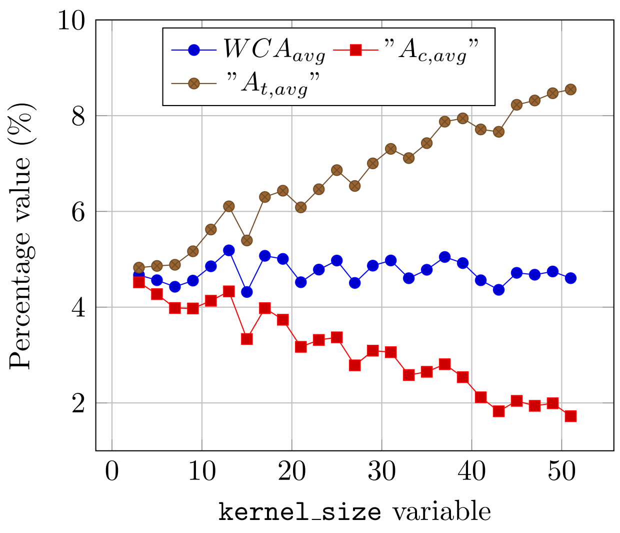


Fig. 6.6: Variation of the percentage areas (relative to the total image area) of WCA_{avg}, $A_{c,avg}$ and $A_{t,avg}$ parameters, increasing the filter size — for the experiment with one camera and the `lane_detector_node`.

The graph presented in Fig. 6.6 shows an increase in the total area of the confidence map ($A_{t,avg}$) as the filter size increases. This is acceptable since black pixels on confidence maps, built through small filters would change to lighter pixels if the filter size increases. The opposite happens for the common area ($A_{c,avg}$) since white pixels transform into darker pixels as the filter size increases. Consequently, the weighted confidence area (WCA_{avg}) remains almost invariable. This is because there is an almost null balance between pixels that are white and become darker (which causes a decrease in this parameter) and those that are black and become lighter (which causes an increase in this parameter), as the filter size increases.

The following study (Fig. 6.7) is related to the variation of the indicators presented at the beginning of this chapter, increasing the filter size.

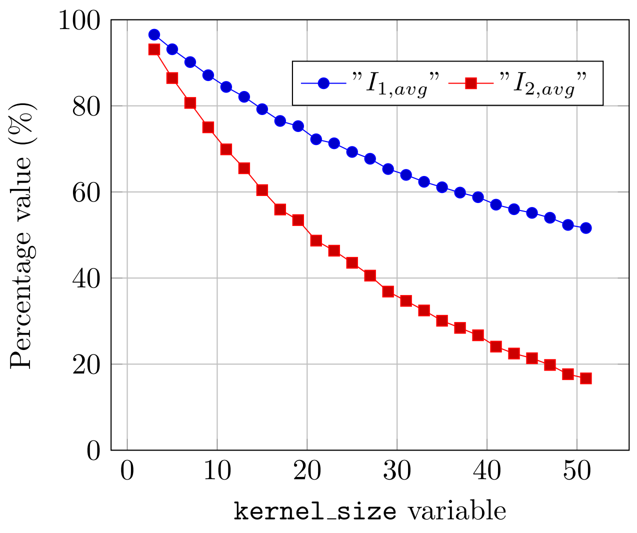


Fig. 6.7: Variation of the average of the indicators increasing the filter size — for the experiment with one camera and the `lane_detector_node`.

In Fig. 6.7 both indicators decrease as the filter size increases but with different rates of variation ($I_{2,avg}$ varies faster in relation to $I_{1,avg}$). This is due to the fact that the first indicator takes into account the weighted confidence area (WCA_{avg}), which is invariant, and the second one is calculated based on the common area ($A_{c,avg}$) that decreases as the filter size increases.

Finally, the values of the standard deviation of each indicator for each filter size are assessed (Fig. 6.8) in order to validate the results presented before.

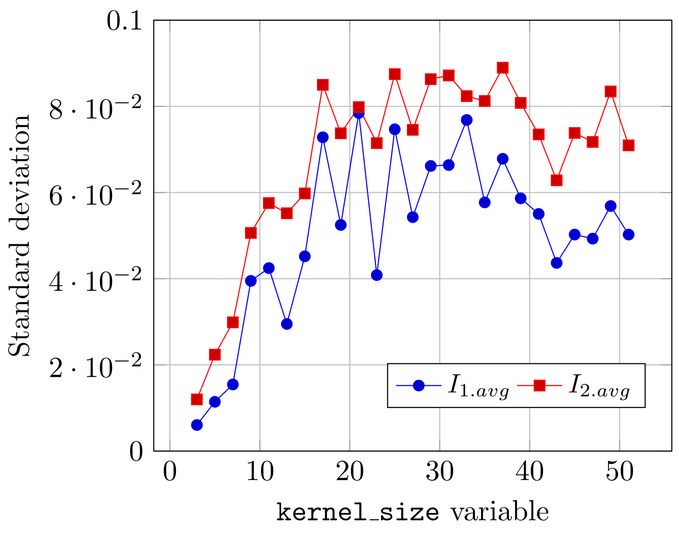


Fig. 6.8: Variation of the standard deviations of each indicator as the filter size increases — for the experiment with one camera and the `lane_detector_node`.

The standard deviation values calculated for both indicators are quite low. This leads to the conclusion that there was enormous proximity between the values obtained for each indicator and the results obtained for this case study are reliable.

6.1.2 Active Processor Node: `advanced_algorithm_node`

Algorithm Behaviour

The second approach concerns the use of the processor "`advanced_algorithm_node`". In terms of practical conclusion, this algorithm achieved regular results, since it demonstrated a solid performance when the real scenario was a straight road (Fig. 6.9).

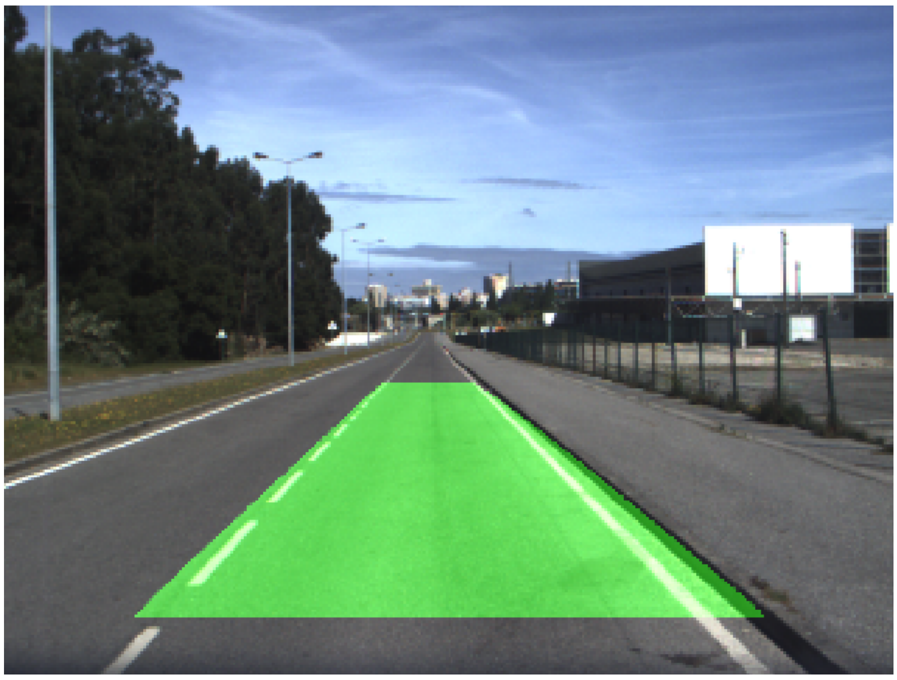


Fig. 6.9: `advanced_algorithm_node` result when facing a straight road.

Moreover, obstacles that occlude the lane lines did not make a difference (Fig. 6.10), which is also a positive point. This algorithm, contrary to the other algorithm used, presents qualitatively average results when the image size is 320 × 240. Consequently, the increase in image size causes an increase in image resolution. This increases the robustness and accuracy of the results obtained by this algorithm.

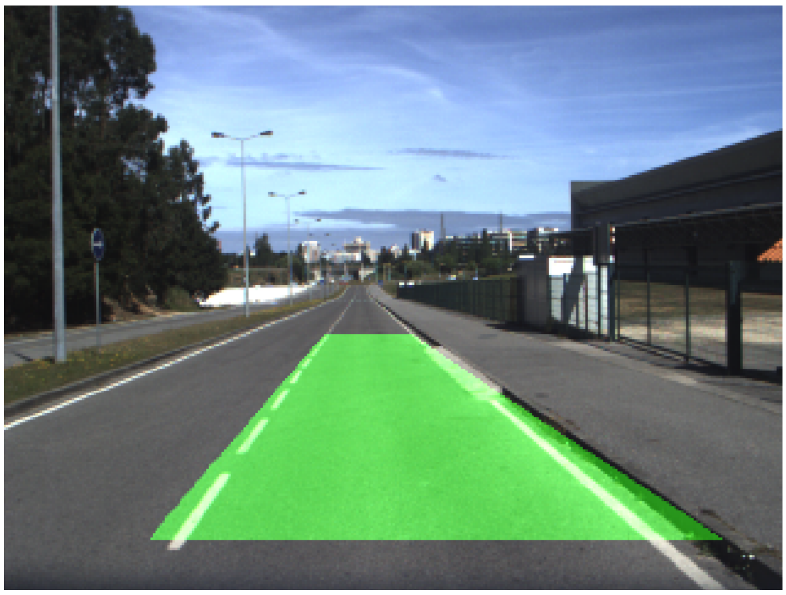


Fig. 6.10: `advanced_algorithm_node` behaviour when facing an occlusion on the right side of the road.

On the other hand, this algorithm presents a particularity that implies to have several precautions when using it — the two lane lines have to intersect the bottom of the image since this algorithm starts to search the white pixels (start of the road lane lines) in the last row of the image (as explained in section 5.2). Thus, the results obtained in circular trajectories such as roundabouts are poor because there is no control over the lane lines in the image (Fig. 6.11).

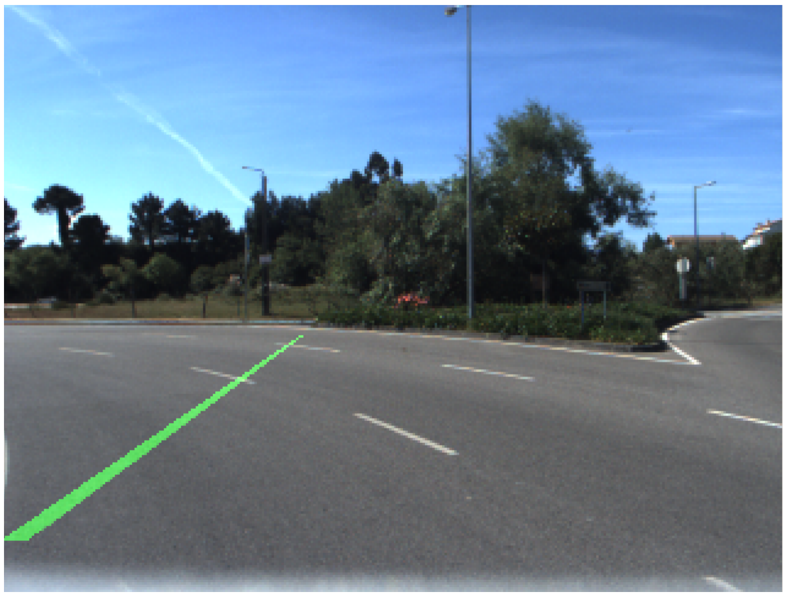


Fig. 6.11: Poor result obtained from the algorithm processing in a particular case — roundabout.

In conclusion, this algorithm is considered a valid algorithm and more optimised than the one presented in section 6.1.1. However, it is not prepared when facing all situations like curved trajectories, so it cannot be considered as an algorithm with excellent performance. The architecture, which is the most important part of the work and the one that has to be evaluated, has been demonstrating a reliable performance in terms of computation. Also, the alteration of the algorithm to be used is made with no problems.

Architecture Results

The size of the filter that promotes the construction of the confidence map is expected to influence not only the effective dimension of the map but also the area that is considered more likely to represent the real roadway. The confidence maps obtained for the two extremes of the filter dimension as well as the algorithm road lane lines detection, are represented in Fig. 6.12.

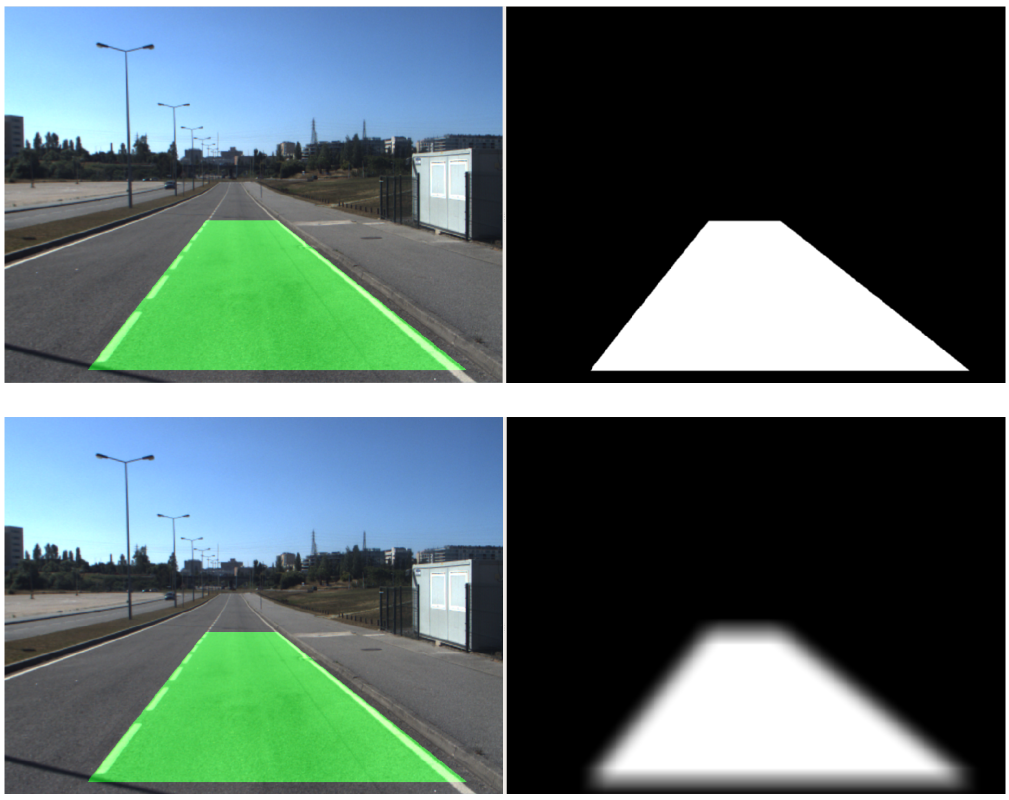


Fig. 6.12: The two images above represent the case of using a filter size of 3 × 3 for the construction of the confidence map. The left image represents the area detected by the algorithm as the road and the right image illustrates the confidence map built from the left image. The images represented below the previous ones have a similar meaning to the one mentioned in the preceding explanation. However, the size of the filter used is 51 × 51. The difference between the filter sizes is found in the blur around the area represented with white pixels on the confidence maps.

The first numerical result presented in Fig. 6.13 is related to the areas used for the calculation of the two indicators explained at the beginning of this chapter. It is expected that the behaviour of these parameters is very similar to the one presented in the previous case (the use of one camera and the `lane_detector_node` as a processor node). It is also expected that in this case, the results obtained would be less dispersed since the polygon detected by the algorithm does not vary as much as in the case of the other algorithm presented.

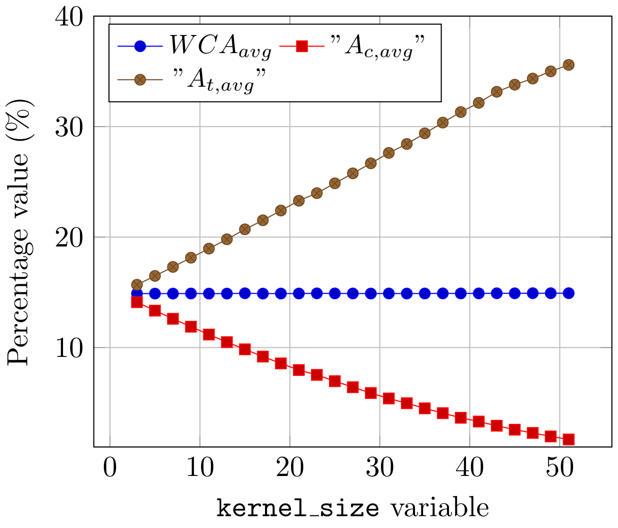


Fig. 6.13: Variation of the percentage areas (relative to the total image area) of WCA_{avg}, $A_{c,avg}$ and A_t parameters, increasing the filter size — for the experiment with one camera and the `advanced_algorithm_node`.

Through the analysis of Fig. 6.13 the following conclusions can be drawn:

- The WCA_{avg} is not related to the size of the filter applied for the construction of the confidence map. This is because although the number of white pixels is decreasing, the number of dark pixels is directly proportional to the filter size. Hence, there is a null balance in the variation of the sum of the value of the pixels of the confidence map;
- The $A_{c,med}$, as expected, decreases as the filter size increases. This is caused by the positive variation of the filter size, which implies that it is more unlikely for all pixels in the neighbourhood of the pixel where the filter is applied, to be part of the navigable zone returned by the algorithm;
- The $A_{t,med}$ is directly proportional to the filter, which would be expected, since a group of pixels that were once black, turn to gray because in its neighbourhood is more likely to have white pixels.

The parameters evaluated for this case compared to those presented in section 6.1.1 are numerically higher, since the detected polygon is larger in the case of the processor `advanced_algorithm_node`.

Consequently, the average indicators ($I_{1,avg}$ and $I_{2,avg}$) are calculated for each filter class evaluated (each class consists of a different filter size). The variation that they suffer due to the increase in filter size is plotted in Fig. 6.14.

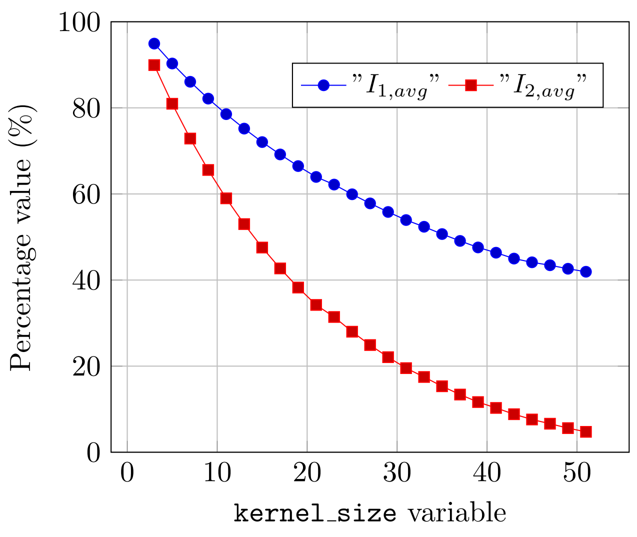


Fig. 6.14: Variation of the indicators as the filter size increases — for the experiment with one camera and the `advanced_algorithm_node`.

Through the analysis of the graph present in Fig. 6.14, it is possible to infer that both indicators are inversely proportional to the filter size. The $I_{2,avg}$, relative to the common area ratio of the confidence map decreases faster since, besides the common area decreases, the total area also decreases. This not occurs to the $I_{1,avg}$, as the WCA_{avg} remains constant. Since the behaviour of the indicators is expected, the architecture used is considered successful in the case of a single camera and one algorithm.

In order to prove that the evaluated samples are homogeneous in terms of dispersion, the standard deviation for each series was also calculated. Its variation throughout each sample is shown in the graph in Fig. 6.15.

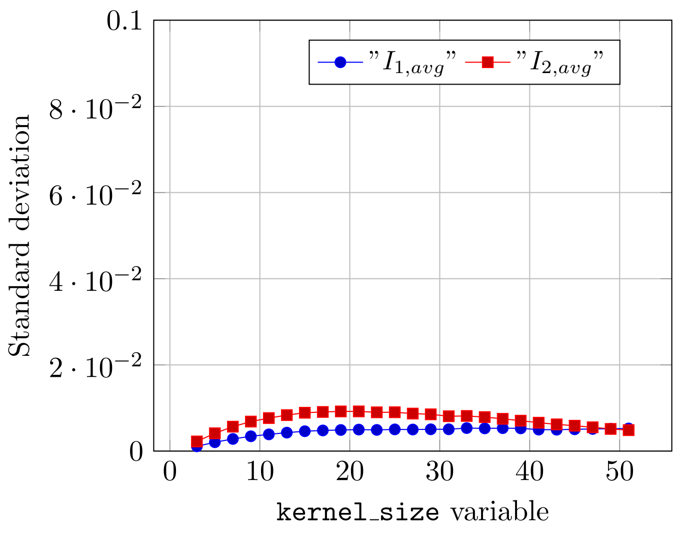


Fig. 6.15: Variation of the standard deviation as the filter size increases — for the experiment with one camera and the `advanced_algorithm_node`.

As can be observed in Fig. 6.15, the standard deviation values obtained are very low, which allows concluding that the samples studied are reliable and homogeneous.

6.2 One Camera and Two Algorithms

This experiment consists of combining the outputs of all available processor nodes. These process an image from a single camera resulting in three distinct results (Fig. 6.16).

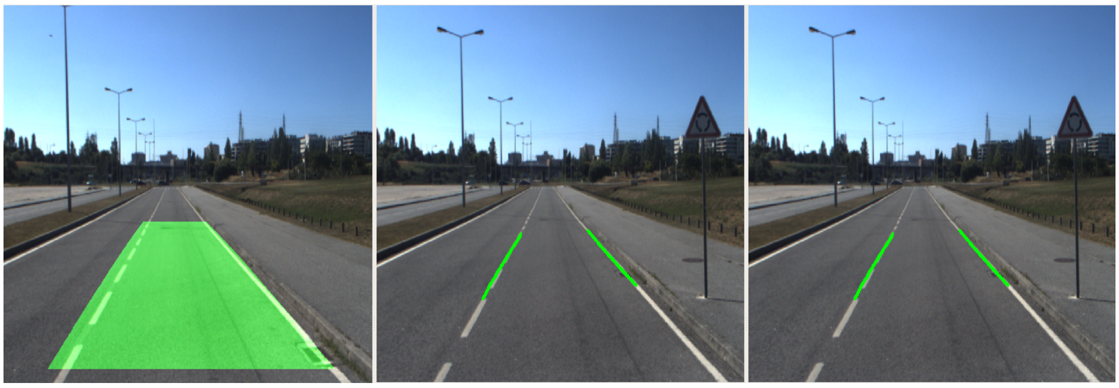


Fig. 6.16: Results obtained from the processor nodes for the case of the use of one camera and two algorithms. The left image represents the result obtained from the `advanced_algorithm_node` that returns a polygon. The centre image is related to the `lane_detector_node` and the right image corresponds to the result obtained by the processor node `lane_detector2_node` (clone of the previous mentioned).

The confidence maps, in this case, have an interesting aspect since the polygon that is the output of one of the algorithms is larger than the polygons that are created by the `draw_poly_node`. Consecutively, the area that is considered as the driveable zone has a smaller area (Fig. 6.17) compared to the total area of the confidence map.

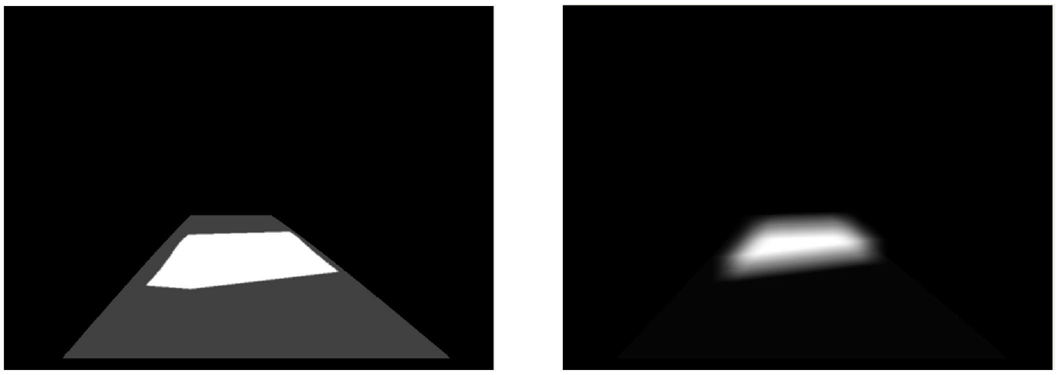


Fig. 6.17: These two images represent the confidence maps created through a dimension of the filter of 3×3 (left image) and 51×51 (right image). It is possible to observe the blur more prominent around the zone that is considered as the driveable zone (white pixels of each image) in the right image.

Another important characteristic of the combination of outputs of the architecture processor nodes that was possible to verify in this experiment is the fact that the area related to less confidence of the confidence map is darker (means less confidence) with the increase of the filter size (detail observed in the Fig. 6.17). This is because as the filter size increases, the lower confidence of the intersection zone (filtered area of intersection of all the polygons provided by each algorithm) decreases. Mathematically, the lowest confidence of the zone of the intersection of those polygons is given by the Eq. 6.6:

$$LC = \frac{\mathrm{ceil}(\frac{side(f_s)}{2})}{f_s} \tag{6.6}$$

Where LC is the lowest numerical value for the pixels confidence of the intersection zone, f_s is the filter size and $(side(f_s))$ is the filter width (e.g. if the filter size is 3×3 then the $(side(f_s))$ is equal to 3 or if the filter size is 25×25 then the $(side(f_s))$ is equal to 25). Therefore as the filter size increases, the numerator increases more than the denominator of the equation that describes the LC.

Regarding the evaluation of the architecture, Fig. 6.18 shows the variation of the constituents of each indicator. It was expected that the architecture would behave differently as it has behaved until now in each of the two previous experiments. That is because of the different relationships between the areas compared to those calculated in the sections 6.1.1 and 6.1.2.

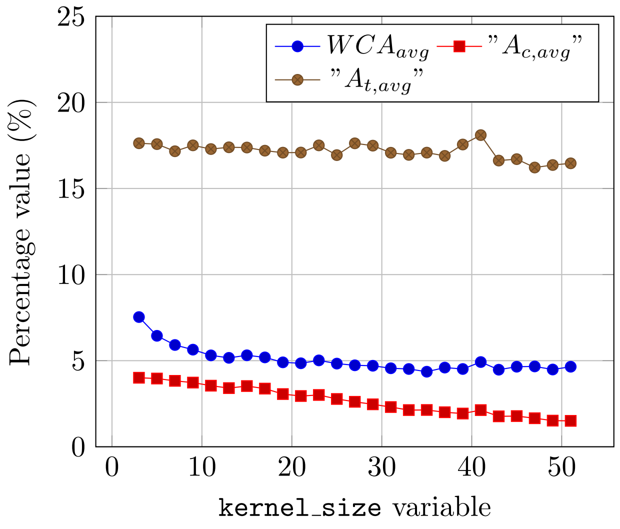


Fig. 6.18: Variation of the percentage areas (relative to the total image area) of WCA_{avg}, $A_{c,avg}$ and A_t parameters, increasing the filter size — for the experiment with one camera and the two processor nodes.

As previously mentioned, this case study is an interesting case in which the confidence map is mostly constituted by low confidence pixels (dark pixels), since the polygon formed by the `/advanced_algorithm_node` is much larger than the polygon that results from the `/lane_detector_node` processing. Consequently, there is just a slight decrease of WCA_{avg}. On the other hand, an invariance in $A_{t,avg}$ is observed, because the higher confidence area is always within the lower confidence zone so what happened for the other experiments (black pixels transformed into lighter pixels), now continues to happen, but these pixels are already within the total area (A_T) of the confidence map. Finally, the $A_{c,avg}$ parameter, as expected, has a behaviour demonstrated until now. So, this parameter decreases with the increase of the filter.

The following analysis concerns the variation of the indicators as the filter size increases (Fig. 6.19).

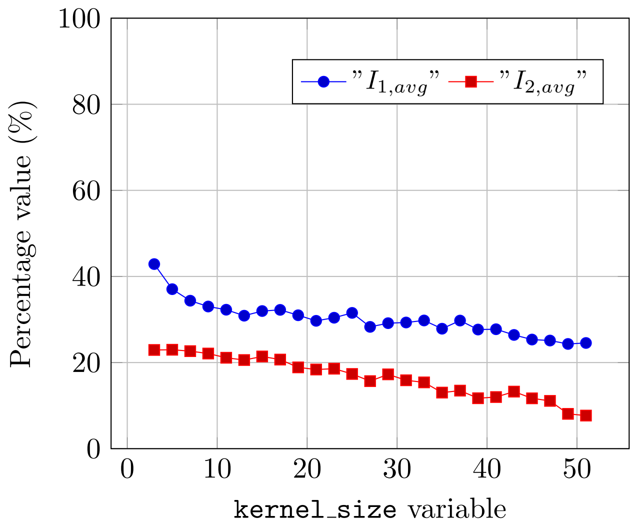


Fig. 6.19: Variation of the indicators as the filter size increases — for the experiment with one camera and the two processor nodes.

The Fig. 6.19 shows that the indicators are inversely proportional to the size of the filter. Based on what was explained in the analysis of the variation of the areas — WCA_{avg}, $A_{c,avg}$ and $A_{t,avg}$ — the indicators vary due to different reasons. In the case of $I_{1,avg}$, it decreases with less intensity compared to previous cases, because of the slight decrease of the parameter WCA_{avg}. On the other hand, the indicator $I_{2,avg}$ decreases due to the natural decrease of $A_{c,avg}$.

Finally, the variation of the standard deviation related to each indicator is shown in Fig. 6.20. Since the variation of the areas was somewhat inconsistent (sometimes showing positive and negative variations for $A_{t,avg}$ and WCA_{avg}), it is expected that the value of the standard deviation for $I_{1,avg}$ is greater than the standard deviation of $I_{2,avg}$.

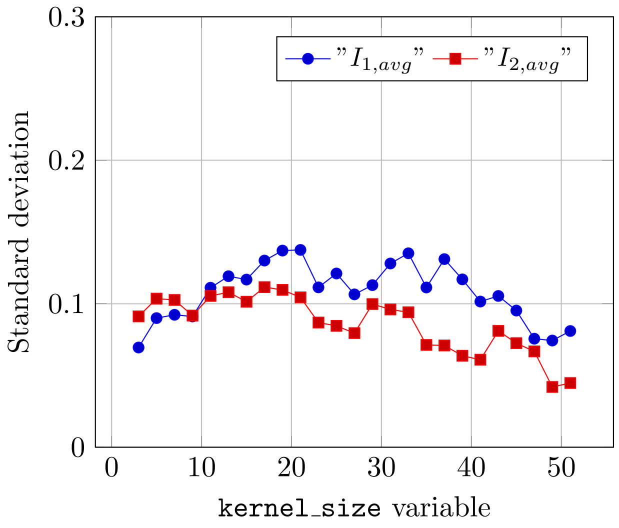


Fig. 6.20: Variation of the standard deviation related to each indicator, increasing the filter size — for the experiment with one camera and the two processor nodes.

By observing the graph presented in Fig. 6.20, what was expected was confirmed. This proves the moderate dispersion of results obtained in this experiment. This can be substantiated by the fact that there is an inconstant variation in the area of the polygon coming from the `lane_detector_node`, due to the unpredictable tracking of the lines. This technique implies that the size of the detected lines is slightly different for the same frame in two launchings of the architecture. The variation of the area of this polygon can affect the indicators that are calculated in each frame of the analysis.

6.3 Two Cameras and One Algorithm

As explained in section 5.3.2, there are two options to launch the architecture in multi-camera mode: with or without a combination of the maps coming from each camera. If the two maps are not combined, the results would be the same as those presented in the previous sections. Therefore, the results presented in this section are based on the combination of the maps coming from each camera.

6.3.1 Active Processor Node: `lane_detector_node`

The study presented in this sub-section concerns the usage of the `lane_detector_node`, which processes images that are captured from two different sources (cameras) — Figs. 6.21 and 6.23. Each algorithm detection implies the creation of a confidence map. In the final stage, the two maps are combined after a perspective transformation, as described in the sections 5.3.2 and 5.3.3.

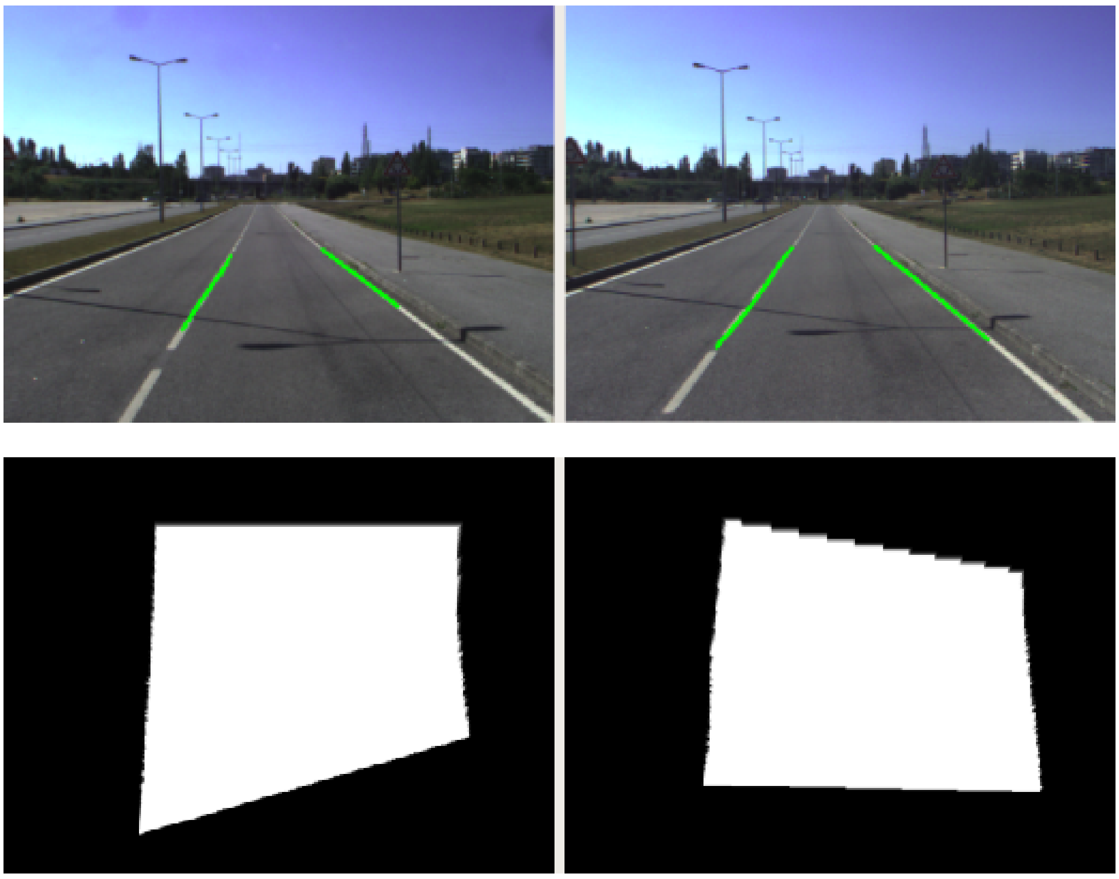


Fig. 6.21: The two images above illustrate the output of the processor node `lane_detector_node` for each camera. The images below represent the confidence map that is built through these road lane lines detection, where the filter applied has a size of 3×3.

After each confidence map is calculated, a final map that combines the previous two is constructed (Fig. 6.22).

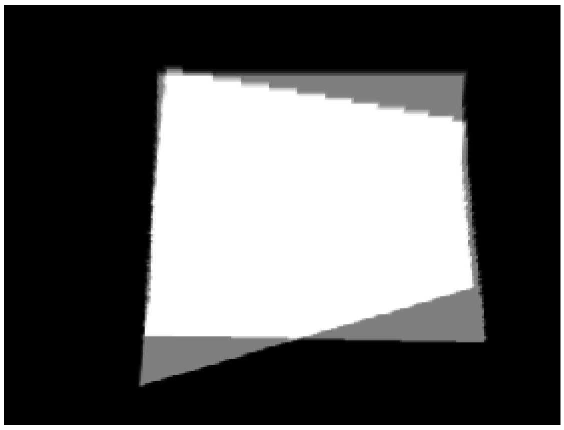


Fig. 6.22: The final confidence map after the weighted sum of the two maps built from each camera.

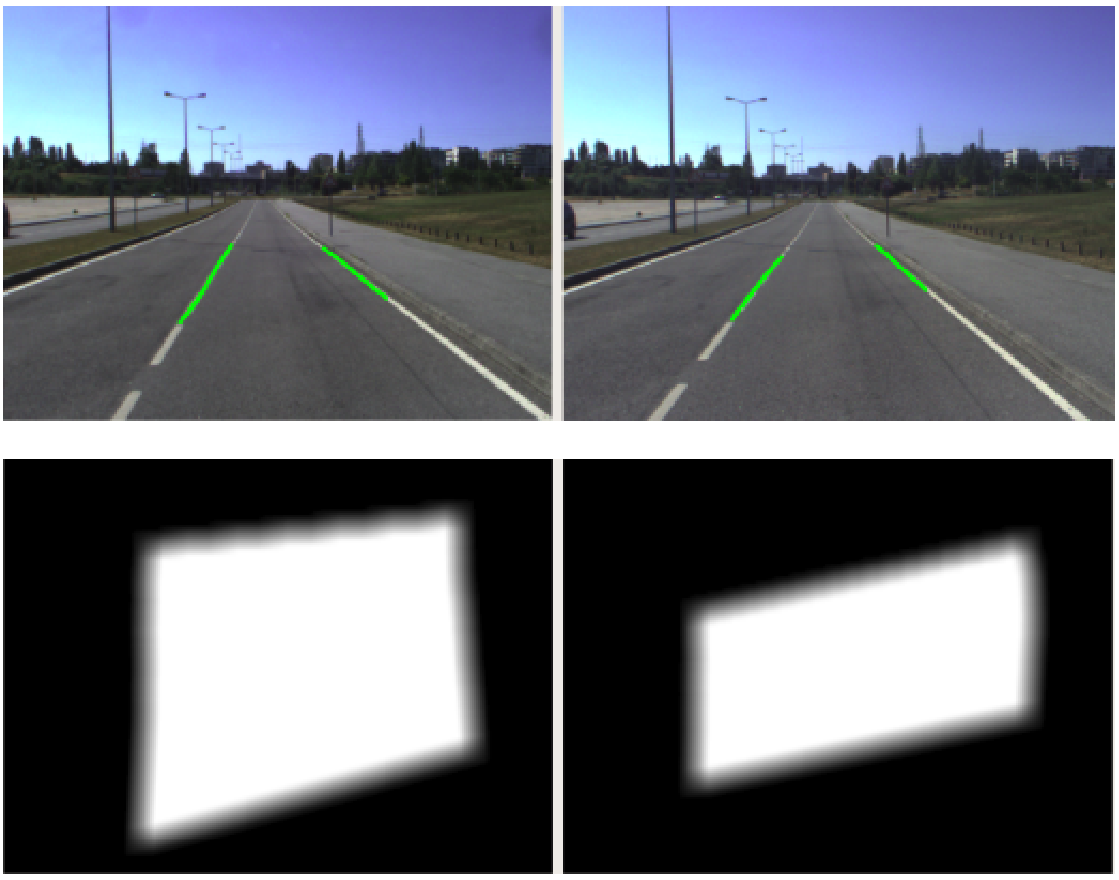


Fig. 6.23: The two images above illustrate the output of the processor node `lane_detector_node` for each camera. The images below represent the confidence map that is built through these road lane lines detection, where the filter applied has a size of 51 × 51.

Finally, the result of the final confidence map is illustrated in the Fig. 6.24.

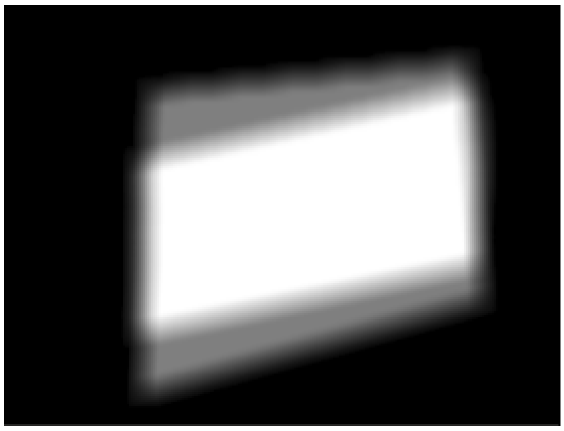


Fig. 6.24: The final confidence map after the weighted sum of the two maps built from each camera.

According to the case study presented in this section, the geometry of the map (rectangular due to the perspective correction that is made) is different than the ones presented so far. This can lead to a smoother application of the filter on the boundaries of the map. Therefore numerical results for the variation of the WCA_{avg}, $A_{c,avg}$ and $A_{t,avg}$ are plotted in the Fig. 6.25.

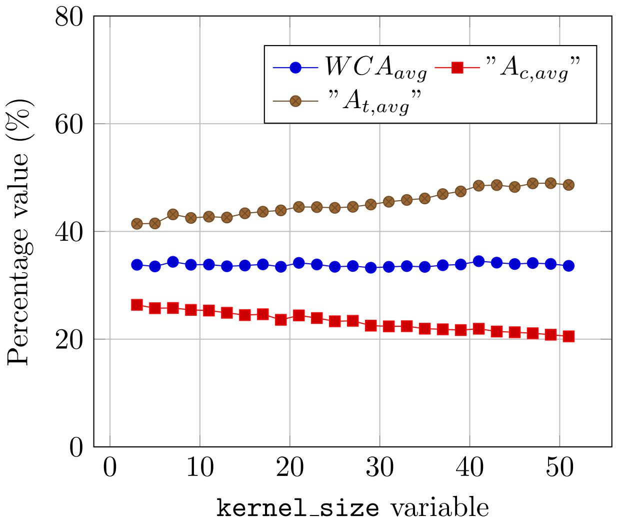


Fig. 6.25: Variation of the percentage areas (relative to the total image area) of WCA_{avg}, $A_{c,avg}$ and A_t parameters, increasing the filter size — for the experiment with two cameras and the `lane_detector_node`.

As can be observed in Fig. 6.25, the variation of all parameters is the expected. Some results affect the linearity of the variation of each parameter. These may be due, once again, to the nature of the algorithm being studied. This nature promotes unpredictability in the size of the polygons that generate the confidence map as mentioned before, which has repercussions on the values obtained for each area and, consecutively, for each indicator analysed (Fig. 6.26).

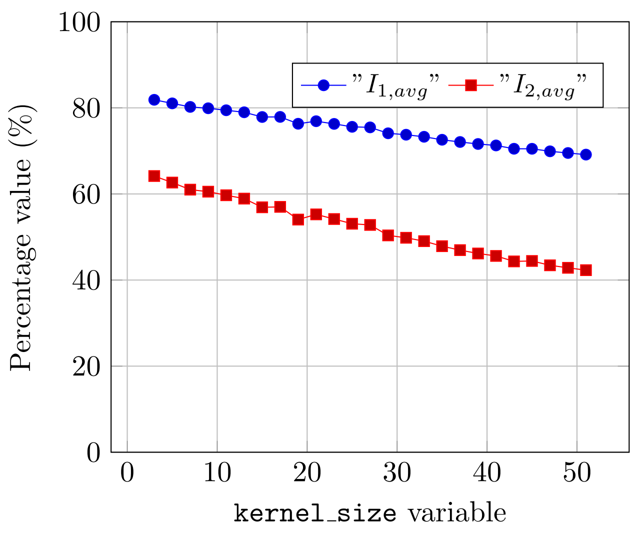


Fig. 6.26: Variation of the indicators as the filter size increases — for the experiment with two cameras and the `lane_detector_node`.

These results show the negative variation of the indicators as the filter size increases. Thus, the results obtained are considered valid. The next analysis (Fig. 6.27) concerns the variation of the standard deviation values increasing the filter size. Here, it is expected that these results would be slightly higher than usual (similar to what happened in the combination of multiple algorithms with one camera) since some results disrespect the linearity of the variation of the indicators.

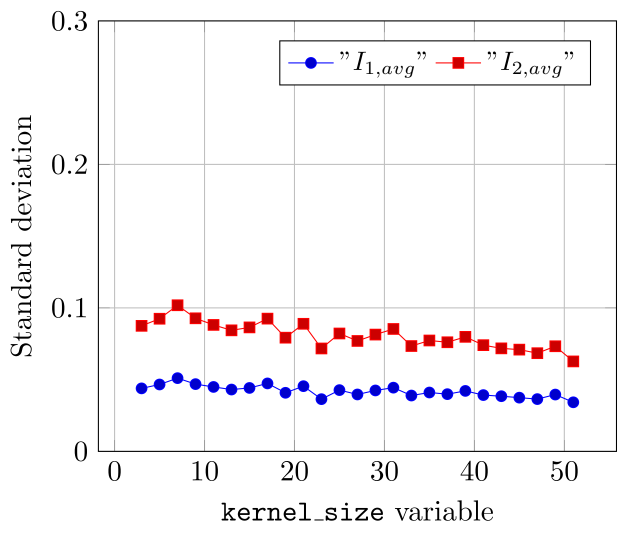


Fig. 6.27: Variation of the standard deviation values related to each indicator as the filter size increases — for the experiment with two cameras and the `lane_detector_node`.

By the observation of the graph represented in Fig. 6.8, some values for the standard deviation are close to 0.1, which shows a small dispersion in the results obtained. This can be due to the same reason as presented before to the parameters $A_{t,avg}$, $A_{c,avg}$ and $A_{t,avg}$.

6.3.2 Active Processor Node: `advanced_algorithm_node`

This experiment is related to the usage of two cameras as sources of images that are processed by the `advanced_algorithm_node`. Similarly to the previous case study (section 6.3.1), the two confidence maps are merged after a warp transformation applied in the polygons. All process is illustrated in the Figs. 6.28 and 6.29.

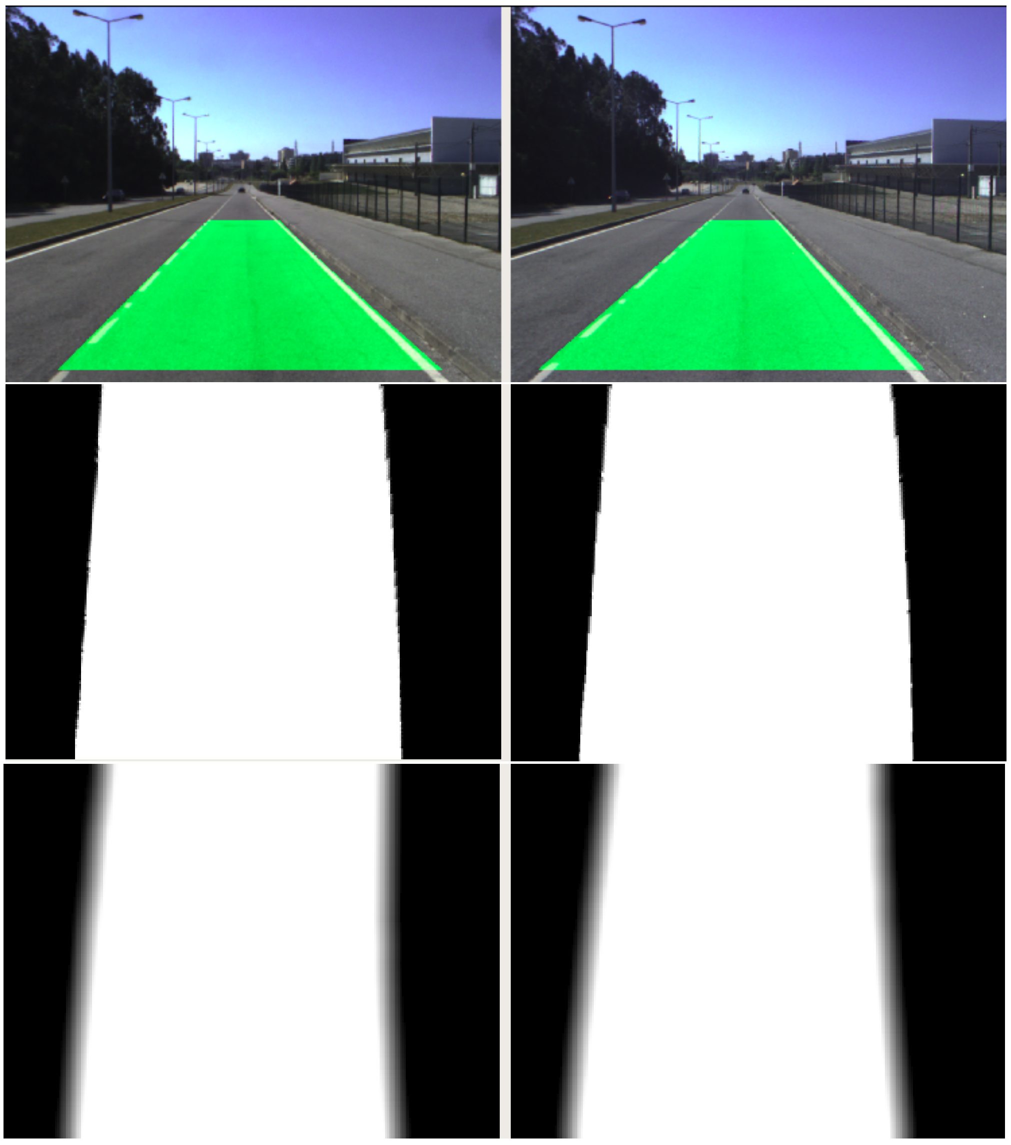


Fig. 6.28: The two images above illustrate the detection of the road area from different perspectives (two cameras with different localisation). The images, below the previous ones, correspond to the confidence map of each camera after a filtering procedure where the filter size is 3×3. The last two images are also confidence maps where the size of the filter applied is 51×51.

Therefore, a confidence map is built by merging each confidence map of each camera resulting in Fig. 6.29.

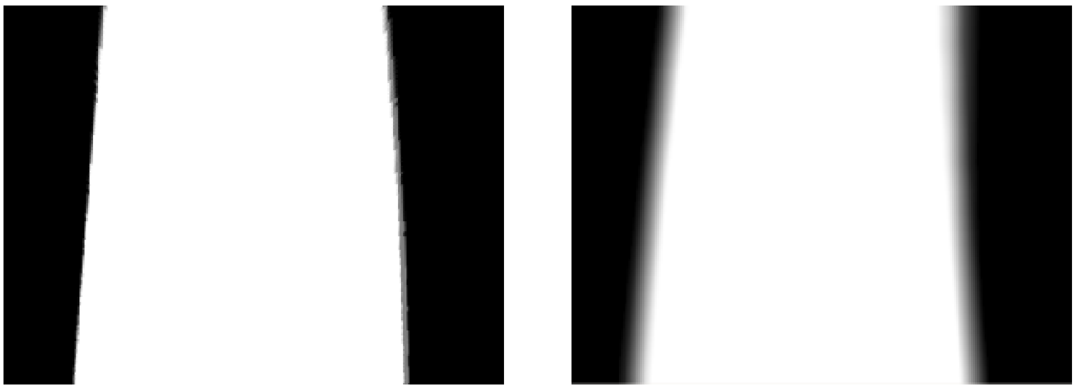


Fig. 6.29: The image represented on the left illustrates the case of a merged map, where the filter applied has a size of 3×3. The right image corresponds to the use of a 51×51 filter.

The confidence map, in this case, is considered more rectangular since the map is from a top-view perspective. Thus, it implies that the blur created by the filtering stage of the map construction is smoother than in the cases where the confidence map corresponds to the camera perspective. Consecutively, the parameters — $A_{t,avg}$ and $A_{c,avg}$ — will vary with a lower rate of change (as occurred in the previous case study). Hence, the first result (Fig. 6.30), shows the variation of the parameters related to the areas as the filter size increases.

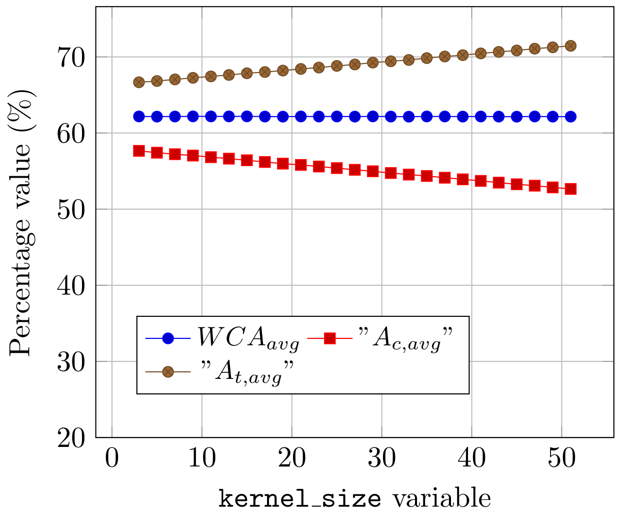


Fig. 6.30: Variation of the percentage areas (relative to the total image area) of WCA_{avg}, $A_{c,avg}$ and A_t parameters, increasing the filter size — for the experiment with two cameras and the `advanced_algorithm_node`.

Through the analysis of the graph presented in Fig. 6.30, it is possible to confirm that the $A_{t,avg}$ and $A_{c,avg}$ parameters have the expected behaviour. The WCA_{avg} parameter is completely constant, which demonstrates the almost null balance between the white pixels that become darker pixels and black pixels that become lighter pixels as verified in the previous sections. The next analysis (Fig. 6.31) consists of the variation of the average indicators with the increase of the filter size.

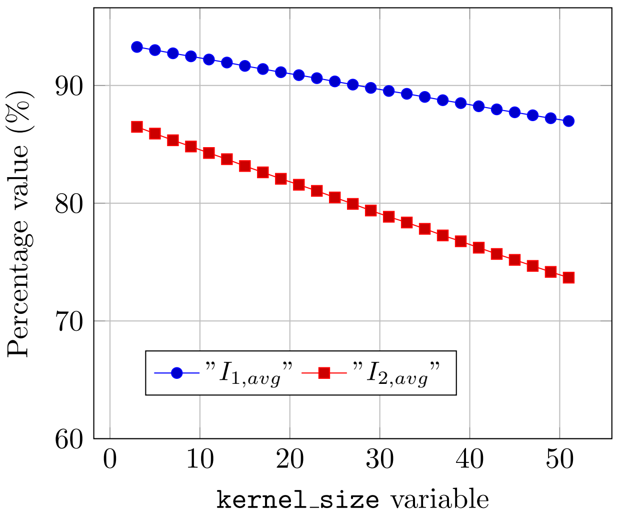


Fig. 6.31: Variation of the indicators as the filter size increases — for the experiment with two cameras and the `advanced_algorithm_node`.

As expected, the indicators suffer a negative variation when the filter size increases. In addition, it is possible to observe that the $I_{2,avg}$ decreases with a higher rate than the $I_{1,avg}$ since beyond the $A_{c,avg}$ decreases, the $A_{t,avg}$ increases. Finally, the results for the standard deviations related to each indicator are shown in Fig. 6.32.

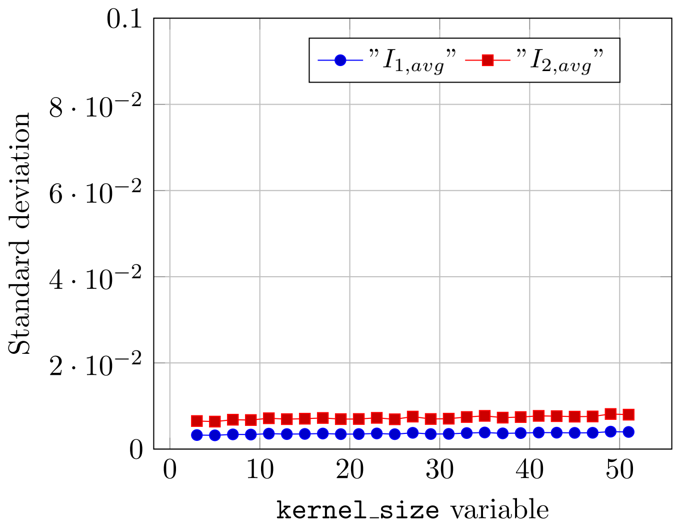


Fig. 6.32: Variation of the standard deviation as the filter size increases — for the experiment with two cameras and the `advanced_algorithm_node`.

As represented in Fig. 6.30, the standard deviations are low, so the sample of the indicators analysed, in this case, is homogeneous, which validates the discussion throughout this case study.

6.4 Two Cameras and Two Algorithms

The last study to be analysed is related to the use of two cameras and two algorithms (Fig. 6.34).

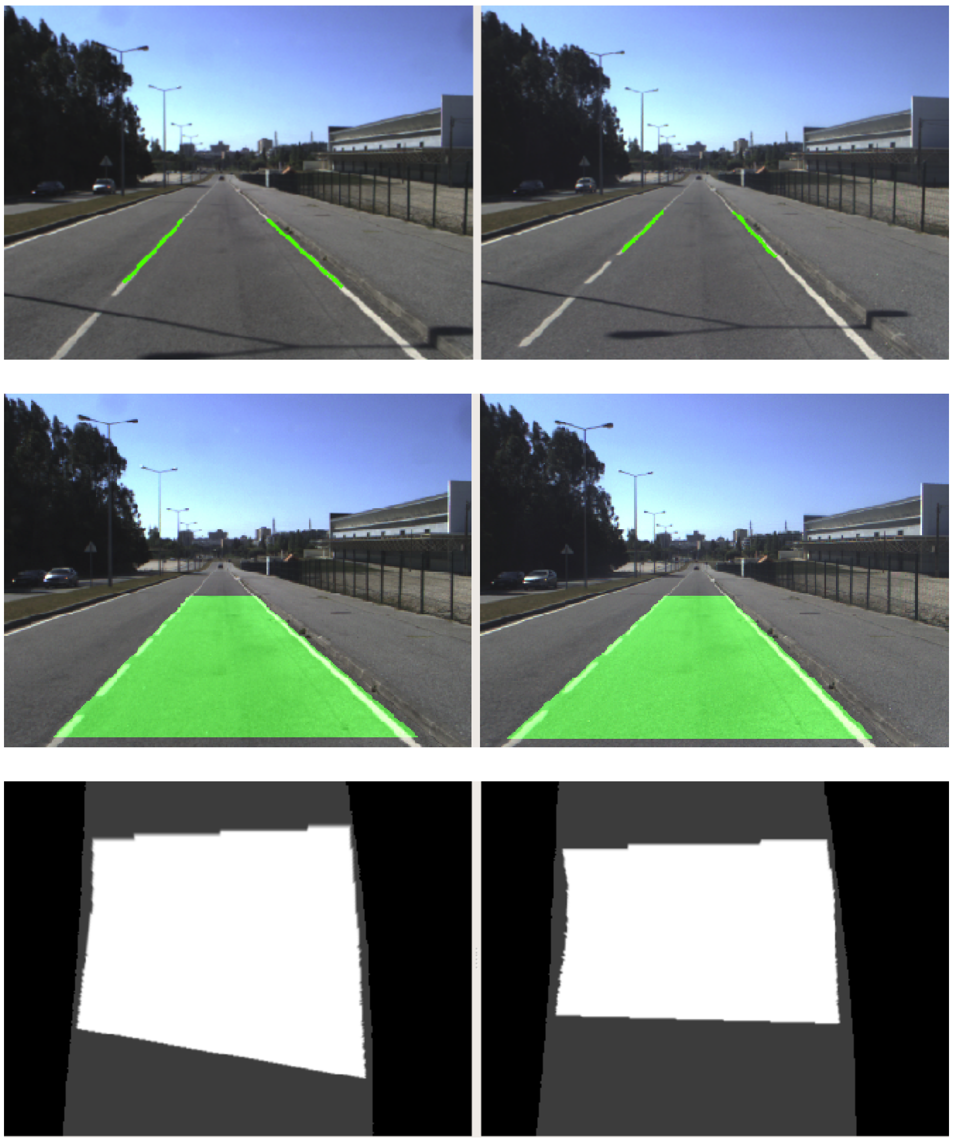


Fig. 6.33: The first two images represent the results obtained by the `lane_detector_node` to each camera (the localisation of the images corresponds to the respective camera perspective). The two images, below the previous ones, correspond to the output of the `advanced_algorithm_node`. Finally, the last two images represent the map after the respective perspective correction.

This experiment represents the maximum exponent of the architecture developed in this work since it incorporates the maximum functionalities available in the architecture - multiple algorithms and multiple cameras. This mode of operation of the architecture promotes the flow of large amounts of information which can lead to a greater overall processing time of the architecture. Also, the final combination may be affected by the amount of data that circulates between the nodes, since a single node — `calc_prob_map_node` — has to process the polygons from different sources and multiple algorithms.

Once the maps of each camera are properly constructed (as has already been demonstrated for the two previous case studies), they are combined in such a way that a single confidence map is constructed (Fig. 6.34).

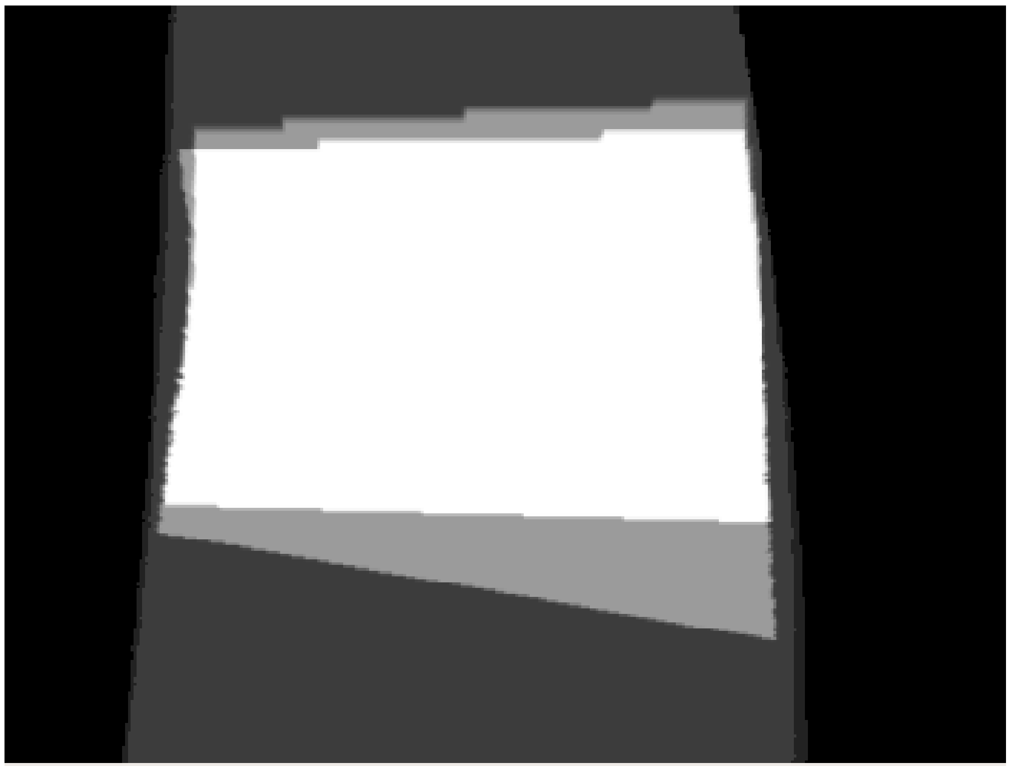


Fig. 6.34: Final confidence obtained by applying a filter whose size is 3×3.

Regarding the results of the architecture, following what happened for the case of using one camera and two algorithms (section 6.2), it is expected that the results obtained by this mode of operation are slightly more heterogeneous (larger standard deviations than those that have been presented for the use of one camera). However, it is expected that the variation of the indicators will be broadly similar to what has been demonstrated so far (decrease of both indicators as the filter size increases). The first result to be discussed is the percentage variation of the areas that constitute the two indicators analysed with the increase in filter size (Fig. 6.35).

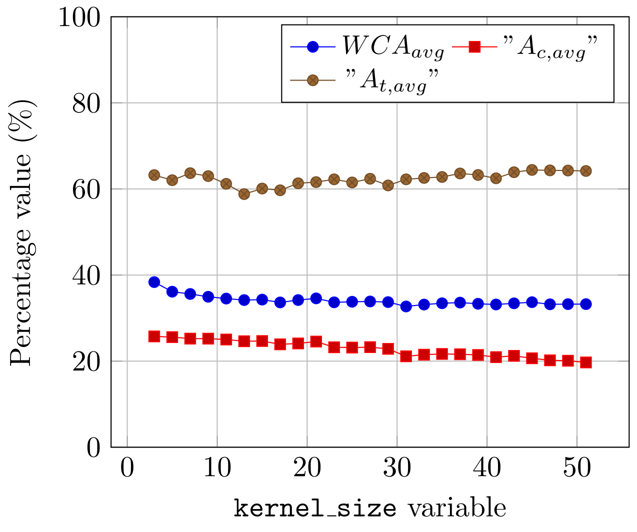


Fig. 6.35: Variation of the percentage areas (relative to the total image area) of WCA_{avg}, $A_{c,avg}$ and A_t parameters, increasing the filter size — for the experiment of two cameras and two processor nodes.

The graph shows a slight increase in $A_{t,avg}$ and decrease in $A_{c,avg}$. This is what should happen under normal architectural conditions, as shown so far. However, similar to the previous case study, there are some unexpected decreases in the variation of $A_{t,avg}$. This may be due, as already explained, to a large amount of data that the computer is processing at the same time. Also, the problems inherent to the algorithm that incorporates the `lane_detector_node` — unpredictable variation of the area of created polygons from the outputs of this algorithm). The following results are those concerning the variation of the indicators as the filter size increases (Fig. 6.36).

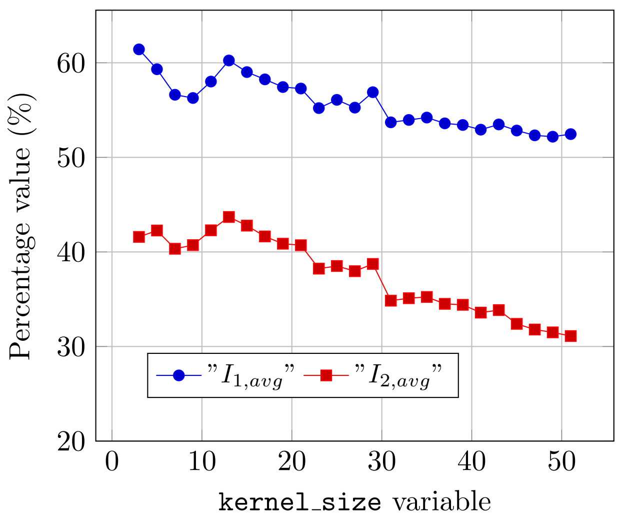


Fig. 6.36: Variation of the indicators as the filter size increases — for the experiment of two cameras and two processor nodes

As can be concluded from the analysis of the graph, the variations of both indicators are overall negative as the filter size increases (expected and valid situation). However, there are some points in the graph that do not follow the trend of the graph's curve. The causes of these "discontinuities" are speculatively similar to those mentioned in the previous analysis. Finally, the analysis of the variation of standard deviations was made to evaluate the homogeneity present in the samples evaluated during this study (Fig. 6.37).

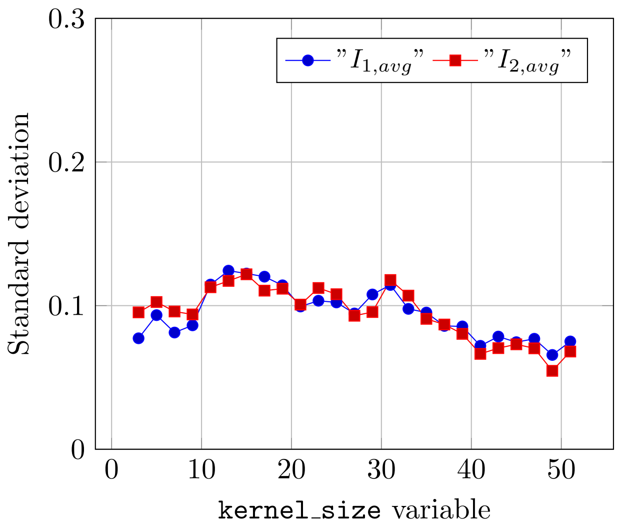


Fig. 6.37: Variation of the standard deviation related to each indicator, increasing the filter size — for the experiment of two cameras and two processor nodes.

Through the analysis of the graph presents in Fig. 6.37, it is inferred that the samples are considered as a whole homogeneous but, as expected, there are some cases of higher standard deviations (greater than 0.1) that represent cases of fluctuations in the values of the indicators that reflect the possible problems that were mentioned at the beginning of this section that may occur in this mode of operation of the architecture.

6.5 Deep Learning Techniques Results

This section presents the results obtained by the two models described in section 5.2.3. These methodologies were applied to the section of the road represented in Fig. 6.38. The first sub-section presents the results obtained for the LaneNet ROS Node and the next sub-section for the UNet Model.

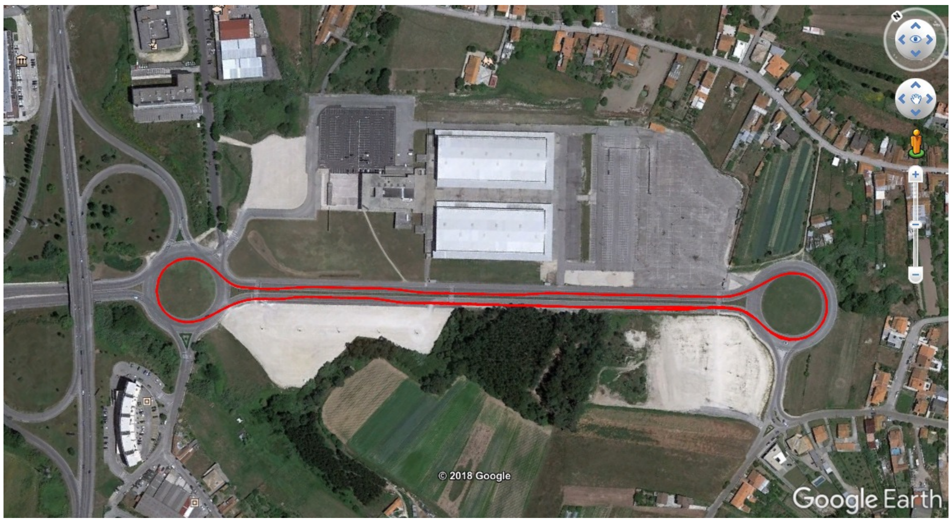


Fig. 6.38: Section of the road that was the subject of the study done for the case of the application of deep learning methodologies (marked in red in the image). This section generated 1070 frames processed by these two methodologies.

6.5.1 LaneNet ROS Node Results

As already stated, this model is in a preliminary stage of development. Thus, there is still a lack of development for curved trajectories (a situation that had been mentioned by the author). Another negative point that the model, currently, presents is the processing time since it does not process the images in real-time yet. However, in qualitative terms, this model is robust and accurate (Fig. 6.39).

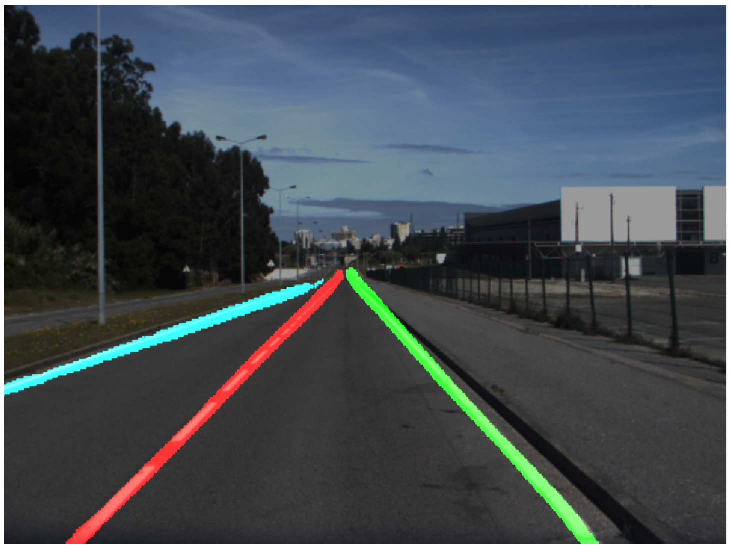


Fig. 6.39: One of the results obtained by the application of the LaneNet ROS node. Besides the precision of the road lane lines detection, it also shows the depth of detection of the road lane lines, which is much higher than any other algorithm performed so far.

The division of lines into instances also works well, however, in certain cases, the discontinuous line is not detected by the algorithm as can be seen in Fig. 6.40. This leads to the conclusion that this ROS node cannot be used at this time on the architecture but can be developed in the future.

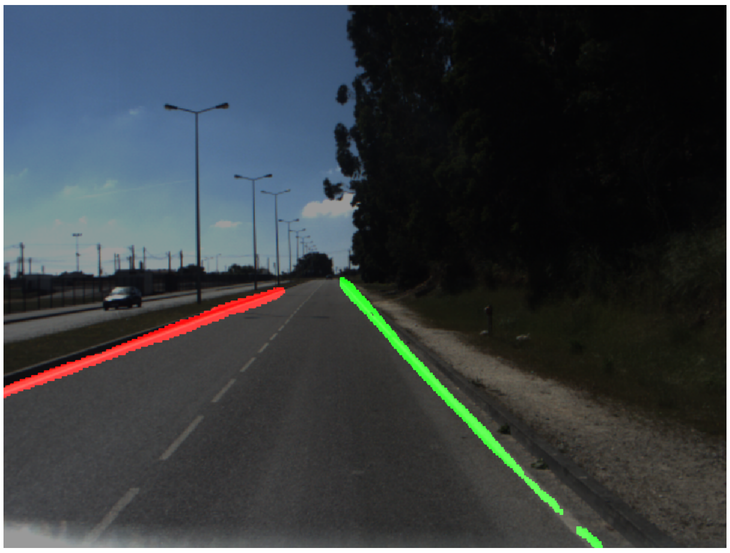


Fig. 6.40: One of the inexact results obtained by this LaneNet ROS Node. Here the discontinuous road line was not detected. Consecutively, the division into instances was imprecise, since the left line that should be represented by the blue colour became represented by the red colour.

In conclusion, this model may be developed in future works. It divides the road lane lines into different clusters, which could have an impact on the type of filters computed during the confidence map construction stage of the architecture. Besides that, this model distinguishes continuous lines of the road (hard obstacles) and broken lines (soft obstacles). This is important information that can be used to formulate a more informed/detailed confidence map.

6.5.2 UNet Model Results

The UNet Model is based on semantic segmentation and was the one that presented the most reliable results of all algorithms used/described so far. Although the training of the neural network was done with only 701 labelled images, the results obtained for the section of road evaluated (Fig. 6.41) were quite satisfying and demonstrate the positive impact that this type of approach can have on this architecture in the future. In terms of this dissertation, the only drawback of this model is that it could not be evaluated in terms of architecture results since it was not a ROS package when this work was in its developmental stage. However, ROS is not yet prepared for such an evolution demonstrating some incompatibilities with Deep Learning approaches. The obstacles faced during the implementation of these methods were: few packages aimed at integrating Deep Learning into ROS projects and a lack of compatibility with the Python 3 version, which offers interesting libraries for Deep Learning implementations (e.g. `fastai` library).

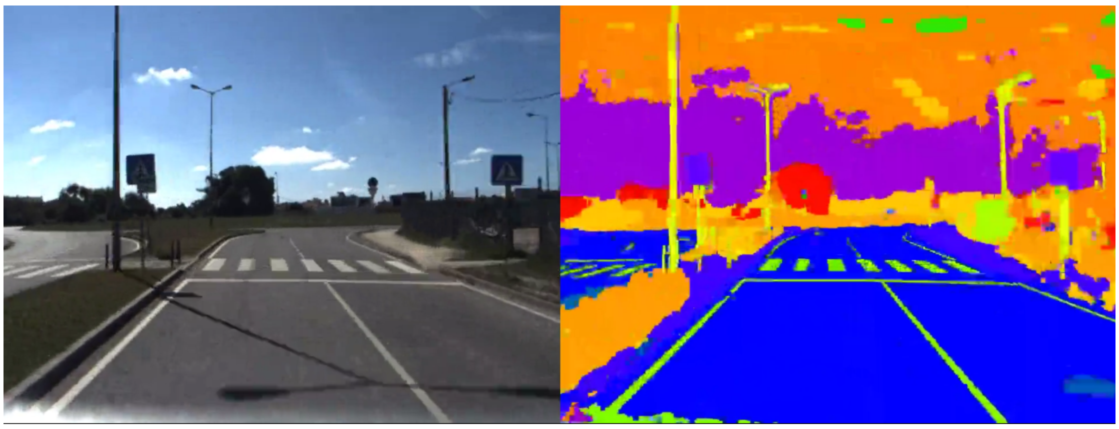


Fig. 6.41: The result obtained by the UNet Model in the case of a crosswalk on a straight road.

The model did not present inaccuracy in any type of trajectory (curve or straight) — shown in Fig. 6.42. This can lead to the idea that if this model were trained with a more generic dataset (a larger number of images in different types of environments), the results would be even better.

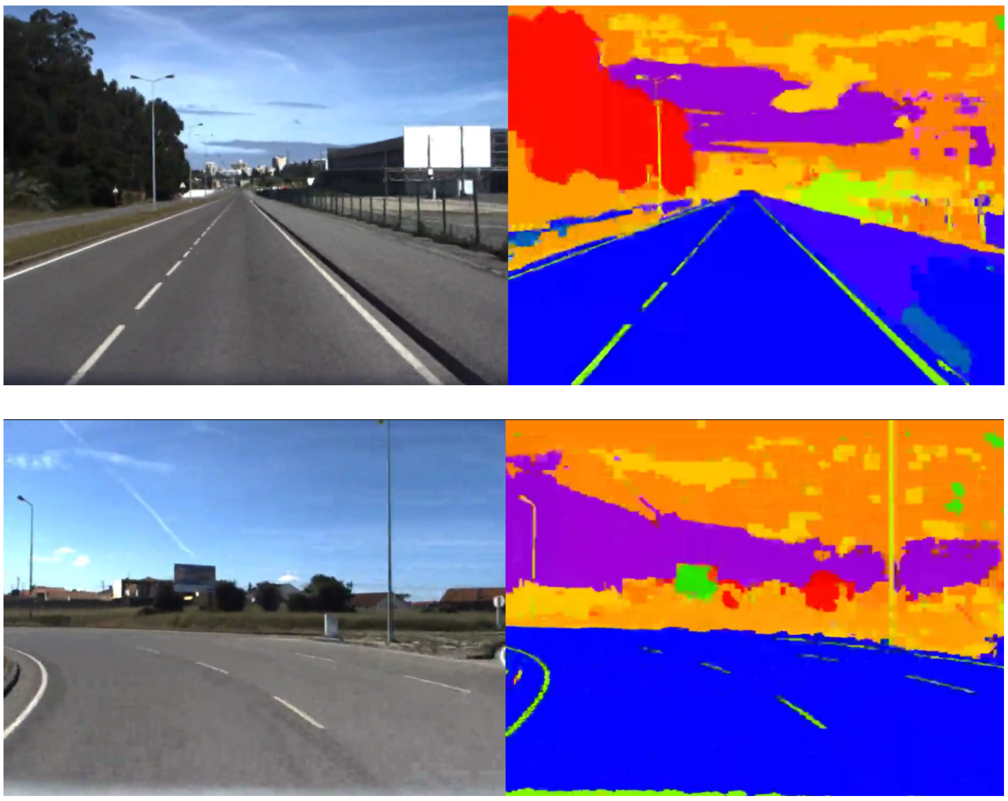


Fig. 6.42: The images shown above correspond to the result obtained by the UNet Model for straight road lanes. The images below are the results obtained by this model for the case of curved trajectories (roundabout).

A future work that would be interesting to develop for application in the architecture already implemented in this dissertation is the implementation of two processor nodes — identical models to those presented in this section (or the use of one that segments the road and the lines of the road at the same time). Therefore, one of the processor nodes would detect road lane lines and the other node would detect the whole road zone (all the lanes included) through semantic segmentation. The node that detects road lane lines would pass through the intermediate stage of the polygon drawing of the architecture to later be intersected with the output of the road semantic segmentation. According to the combination stage of the architecture, the intersection would be the area that would be associated with a 100 % confidence in being considered as a road lane and the remainder that would be given in this case by the road segmentation would be less likely to be considered a road. This situation is interesting since the lane road of the road where the vehicle circulates would be given as a 100 % driveable zone and the lane roads around it would have less confidence of the vehicle circulating there. In conclusion, a frame-based confidence map of the road would be created, with much more information than if the algorithms were used individually as it is happening in the actual stage of the architecture but with less reliability. Two samples that can serve as examples of the case are represented in Figs. 6.43 and 6.44.

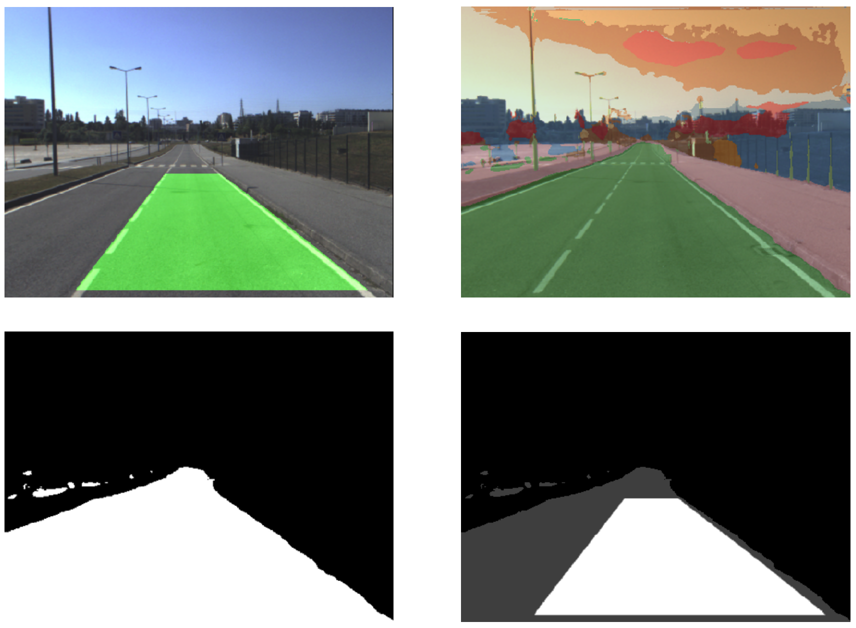


Fig. 6.43: This set of images represents the combination of classic techniques and modern approaches achieved at the final stage of this dissertation. The upper left image is the output of the `advanced_algorithm_node`. The upper right image is the output of a UNet model ROS node. This node was developed to study the potential combination of two types of computer vision techniques: traditional (first image) and modern (second image). Thus, the bottom left image corresponds to the input in the combination step of the architecture explained in this work. Finally, the last image represents the combination of the two types of techniques in one single confidence map (the filter applied in this case has a size of 3×3).

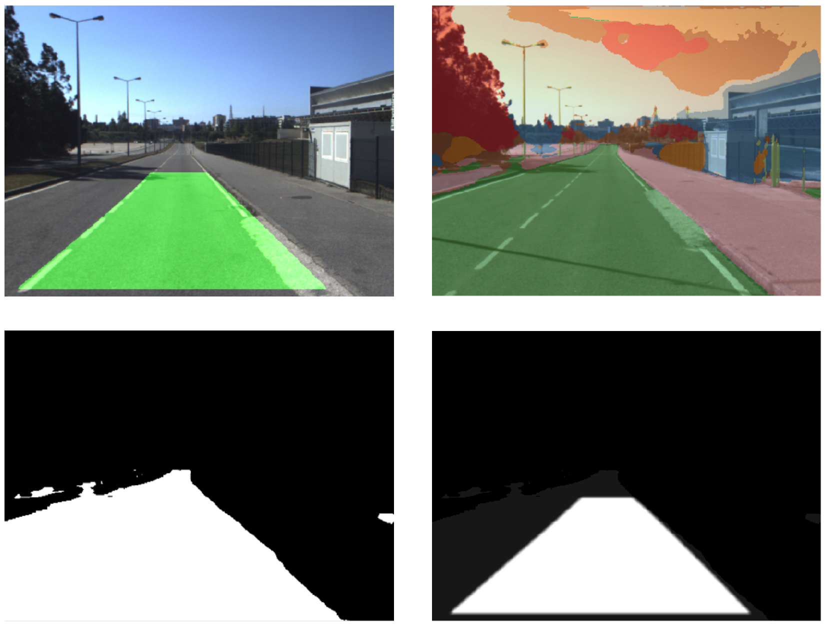


Fig. 6.44: This is a similar case to the one presented in Fig. 6.43. However, here the size of the filter is larger ((9×9)). Consecutively, it can be observed that less confidence is given to the pixels belonging to the area of the other lane, as would be expected.

As can be observed through the analysis of Figs. 6.43 and 6.44, the confidence map shows the maximum confidence in the intersection zone between the two different algorithms (white pixels in the last image). However, lower confidence is given to the other lane of the road. This may be interesting for future decisions of the vehicle planner, since the road on where the vehicle is driven means greater confidence for the vehicle to be driven in the future and the lane next to it (where it can also be driven) is associated to lower confidence, which may mean that the vehicle can be driven there with the necessary precautions.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In the fields of autonomous driving and ADAS, the visual perception of the road boundaries is an important feature that has been studied and developed to join information about the surroundings of the vehicle. This data allows avoiding collisions since the car perceives the lane where it is travelling on. The work presented in this document is the development of an architecture capable of receiving images captured by multiple cameras and processed later by multiple algorithms.

There are several different types of algorithms for road lane detection. However, the most common types of algorithms' output are the spatial localisation of the road lane lines in the image and a polygon that represents the navigable zone of the road. Consequently, the deployed architecture is prepared to receive these two different types of algorithms' results. The integration of different types of algorithms in a single architecture could be susceptible to long processing time. Hence, the core of the developed architecture (a combination of the algorithms' results) is based on simple computer vision techniques, such as logical operations "AND" and "XOR" and image filtering. A road confidence map is built, through the application of these techniques. This map is an image and enables the knowledge of the navigable zone divided into areas of greater or lesser confidence in belonging to the correct representation of the road lane. Since the domain of the architecture application is the road lanes/boundaries detection, it depends on the algorithms that are computed to detect the road lane lines or the road boundaries. Then, two algorithms based on classical computer vision techniques were implemented. One of them was already a ROS package. The other algorithm was successfully transformed into a ROS package, based on an adaptation of a code developed in a course context about autonomous driving. It demonstrated more accuracy and less processing time than the previously mentioned. However, none of them obtained consistent and reliable results to belong to the developed architecture in the future because they do not present valid qualitative results in different types of environments. Conversely, modern algorithms based on Deep Learning were also computed and the results obtained are much better than the ones presented for the previous algorithms, which are based on classical techniques of computer vision. To conclude, the modern algorithms have to be developed and studied in order to be part of the processor nodes of this architecture and possibly combined to the traditional computer vision techniques.

Regarding the multi-camera mode that the architecture offers, there is the possibility that the maps from each camera can be combined using an IPM technique. The technique developed in this work is not generic since the width of the road is considered known previously. However, the architecture also enables the launch of multiple cameras without combining the maps.

In addition to the architecture developed, some hardware was installed in the ATLASCAR2 such as two Pointgrey FL3 GE28S4-C, two protective boxes to the cameras and the respective fixation systems, as well as all the operative material that enables the cameras to operate appropriately.

Finally, the combination of classical computer vision techniques with modern techniques is an interesting breakthrough in these studies related to the visual perception applied to autonomous driving. Besides that, the developed architecture promotes a result that gathers more information than any algorithm running individually. Hence, an article, named "Scalable ROS-Based Architecture to Merge Multi-source Lane Detection Algorithms",was submitted to the *Robot2019* Conference authored by Tiago Almeida, Bernardo Lourenço and Vítor Santos.

The code of this work is presented and documented on the Github platform (`http://github.com/lardemua/road_visual_perception`).

7.2 Future Work

Future work has to include the development of the algorithms that are part of the processing stage of the architecture. They are the infrastructure that provides the practical results in terms of road lane lines or boundaries detection. Since the measure of these is lacking in this work (because the architecture is the part that was developed and treated carefully and not the algorithms), it would be interesting to measure the difference between the algorithms applied individually and combined by the implemented architecture. It is also important to include another type of algorithms' output on the architecture to turn the architecture even more generic.

The phase that consists of the multiple algorithms combination can include a multi-level confidence degree based on the number of algorithms that intersect the same zone. The work developed in this document does not take into account how many algorithms return the same zone. The current combination of the algorithms only results in a maximum confidence to zones that are the intersection of all algorithms and gives a lower confidence to all other zones (which can correspond to the output of one algorithm or the intersection of $n - 1$ algorithms, where n is the number of algorithms used to process the images captured by the camera).

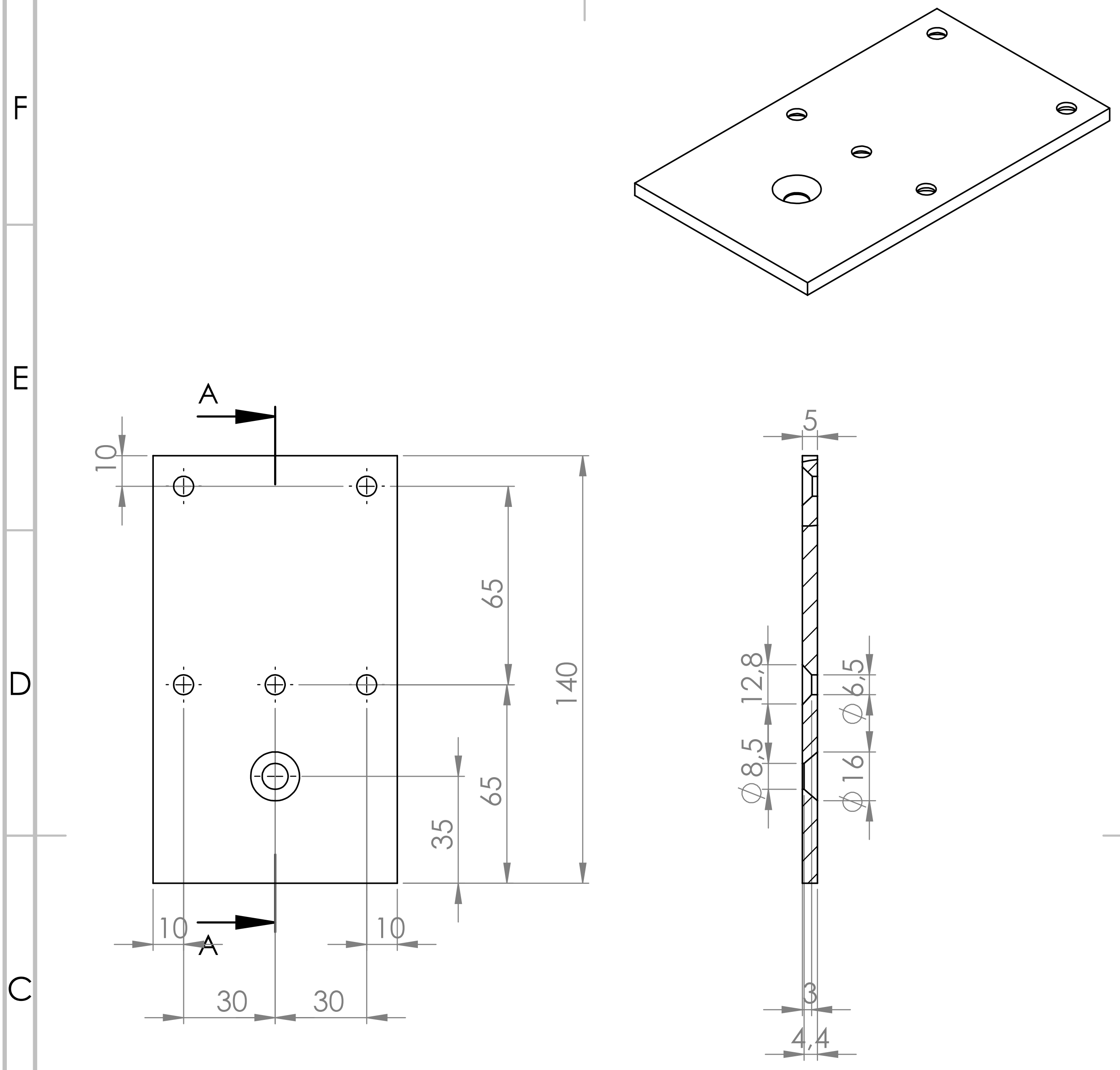
In terms of extrinsic calibration, it is crucial to do an accurate calibration based on the work developed in [48] to reduce the orientation errors, since even a one-degree error translates to meters at large distances.

Regarding the architecture output, many minor improvements can be implemented; however, there is also a high priority feature that needs to be deployed: the application of the IPM technique to the final confidence map (based on an accurate extrinsic calibration). It can be published as an Occupancy Grid to merge it with the data that came from the LIDAR sensor [49]. If this were achieved, the ATLASCAR2 would base its decisions in a synchronous Occupancy Grid (navigation map). It would fuse information from two different types of sensorial sources (LIDAR and cameras).

Intentionally blank page.

Appendix A

Plate to fix the box to the rotative support



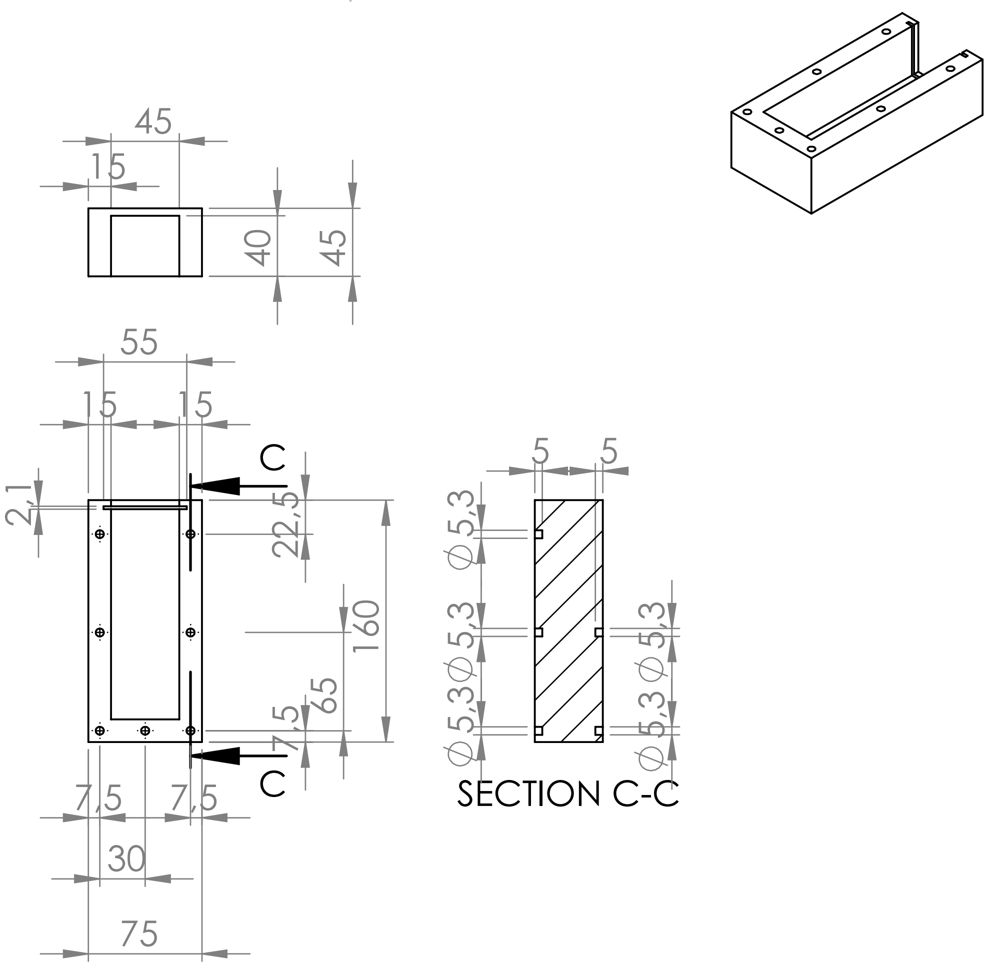
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR: ISO 2768-m	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING / REVISION

	NAME	SIGNATURE	DATE		TITLE:
DRAWN					
CHK'D					
APPV'D					
MFG					
Q.A				MATERIAL:	DWG NO. chapa A4
				WEIGHT:	SCALE:1:2 SHEET 1 OF 1

SOLIDWORKS Educational Product. For Instructional Use Only.

Appendix B

Box to protect the camera



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: ISO 2768 -m LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION

	NAME	SIGNATURE	DATE			
DRAWN						
CHK'D						
APPV'D						
MFG						
Q.A						

TITLE:

MATERIAL:

DWG NO.

caixa_imp_final

A4

WEIGHT:

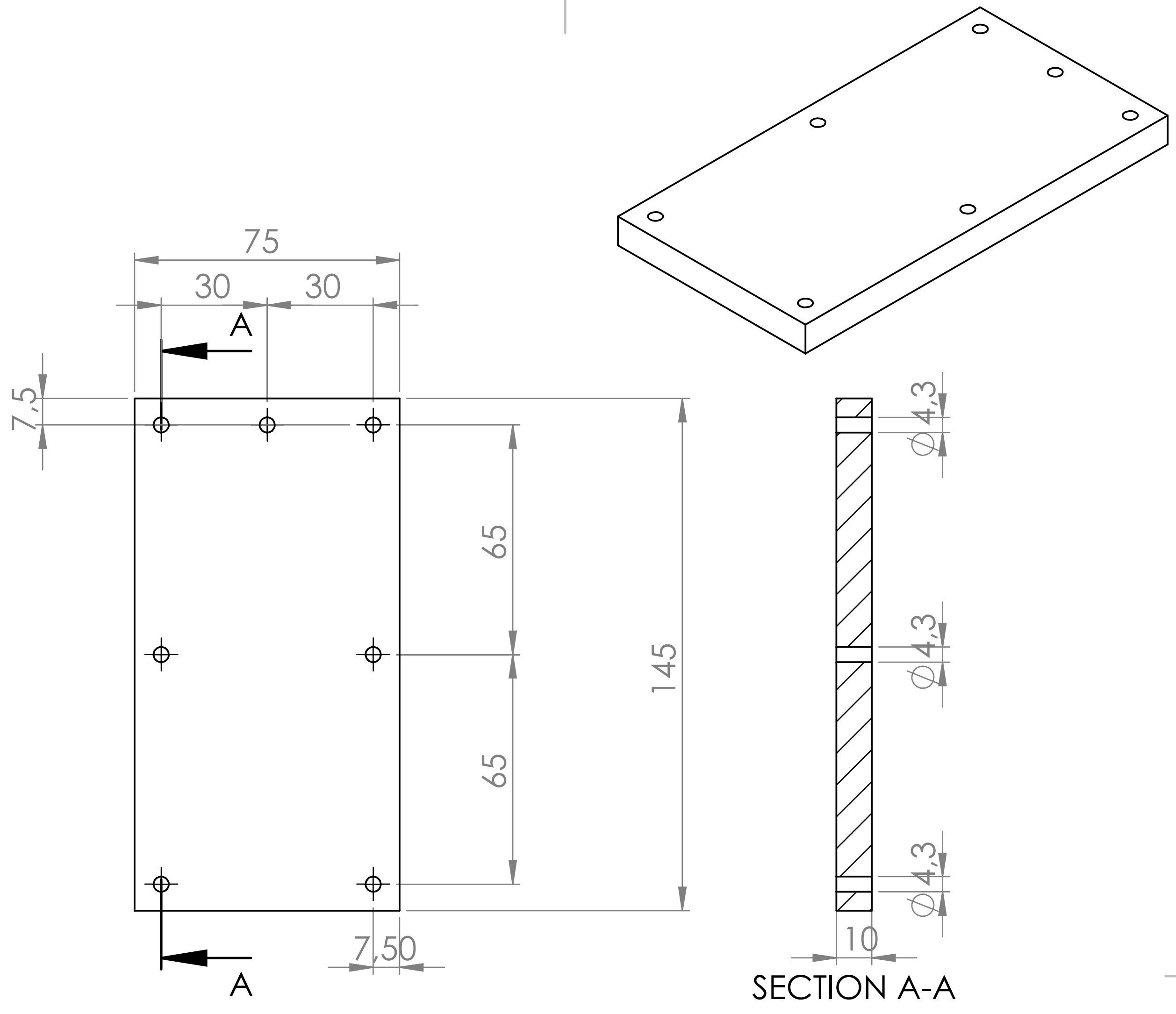
SCALE:1:5

SHEET 1 OF 1

SOLIDWORKS Educational Product. For Instructional Use Only.

Appendix C

Box's carton



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: ISO 2768-m LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING

REVISION

	NAME	SIGNATURE	DATE
DRAWN			
CHK'D			
APPV'D			
MFG			
Q.A			

TITLE:

MATERIAL:

DWG NO.

tampa_caixa_final

A4

WEIGHT:

SCALE:1:2

SHEET 1 OF 1

SOLIDWORKS Educational Product. For Instructional Use Only.

References

[1] Machine vision for self-driving cars – current applications. https://emerj.com/ai-sector-overviews/machine-vision-for-self-driving-cars-current-applications/. Accessed: 2019-02-15.

[2] Su CY. and Fan GH. An effective and fast lane detection algorithm. in: Bebis g. et al. (eds) advances in visual computing. *Lecture Notes in Computer Science*, **5359**:942–948, 2008.

[3] Atlas project. http://atlas.web.ua.pt/. Accessed: 2019-02-21.

[4] Abdulhakam.AM.Assidiq. Real time lane detection for autonomous vehicles. In *Intelligent Vehicles Symposium*. IEEE, 2008.

[5] A brief history of lane departure warnings. https://medium.com/@ducannissan/a-brief-history-of-lane-departure-warnings-f6316fce8427 . Accessed: 2019-02-25.

[6] Mitsubishi debonair. https://carfromjapan.com/specifications/mitsubishi/debonair-v/581789df2afaa2c4b2874e3f. Accessed: 2019-02-25.

[7] Honda worldwide - world news - news release. https://web.archive.org/web/20141230004825/http://world.honda.com/news/2003/4030618_2.html . Accessed: 2019-02-21.

[8] tesla model s adds 'speed assist', lane-departure warning. https://www2.greencarreports.com/news/1094773_tesla-model-s-adds-speed-assist-lane-departure-warning . Accessed: 2019-02-21.

[9] Toyota crown majesta. https://en.wikipedia.org/wiki/Toyota_Crown. Accessed: 2019-02-25.

[10] Paul C. Hough V. Method and means for recognizing complex patterns, December 1962.

[11] Roshan Jahan, Preetam Suman, and Deepak Kumar Sing. Lane detection using canny edge detection and hough transform on raspberry pi. *International Journal of Advanced Research in Computer Science*, pages 85–89, 2018.

[12] Tesla model s. https://www.zigwheels.com/newcars/Tesla/model-s. Accessed: 2019-02-25.

[13] Volvo cars announces range of updates for model year 2017. `https://www.media.volvocars.com/global/en-gb/media/pressreleases/175764/volvo-cars-announces-range-of-updates-for-model-year-2017` . Accessed: 2019-02-25.

[14] Lane keeping assist system (lkas) (honda sensing® models). `https://www.hondainfocenter.com/en/Civic-Family/2018-Civic-Hatch/Product-Details/Interior/Lane-Keeping-Assist-System-LKAS-Honda-Sensing-models` . Accessed: 2019-03-05.

[15] Ricardo Morais. *Parametrização de algoritmos para classificação de estrada a bordo do ATLASCAR*. Master thesis, University of Aveiro, 2014.

[16] Mohamed Aly. Real time detection of lane markers in urban streets. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 7–12. IEEE, 2008.

[17] Amol Borkar, Monson Hayes, and Mark T. Smith. Robust lane detection and tracking with ransac and kalman filter. pages 3261–3264, 11 2009.

[18] Miguel Oliveira, Vitor Santos, and Angel Sappa. Multimodal inverse perspective mapping. *Information Fusion*, 09 2014.

[19] Tutorial: Build a lane detector. `hhttps://towardsdatascience.com/tutorial-build-a-lane-detector-679fd8953132`. Accessed: 2019-02-21.

[20] Udacity advance lane-detection of the road in autonomous driving. `https://medium.com/deepvision/udacity-advance-lane-detection-of-the-road-in-autonomous-driving-5faa44ded487`. Accessed: 2019-02-20.

[21] K. Kluge and S. Lakshmanan. A deformable-template approach to lane detection. *IEEE*, pages 54–59, 1995.

[22] Y. Wang, E. Teoh, and D. Shen. Lane detection using b-snake. *Int. Conf. Information Intelligent and Systems*, pages 438–443, 1999.

[23] Xiadong Miao, Shunming Li, and Huan Shen. On-board lane detection system for intelligent vehicles based on monocular vision. *International Journal on Smart Sensing and Intelligent Systems*, **5**(4):957–972, 2012.

[24] Lane_detector ROS package. `https://github.com/Nsteel/Lane_Detector`. Accessed: 2019-02-18.

[25] Tejus Gupta, Harshit Sikchi, and Debashish Charkravarty. Robust lane detection using multiple features. *IEEE Intelligent Vehicles Symposium (IV)*, 2018.

[26] FLIR ® Inc. *Flea3 GigE Techincal Reference*, 2017. Rev. 8.0.

[27] Carol Fairchild and Dr. Thomas L. Harman. *ROS Robotics By Example*. PACKT, 2016.

[28] image_proc ros package. `http://wiki.ros.org/image_proc` . Accessed: 2019-02-15.

[29] Adrian Kaehler and Gary Bradski. *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library.* O'Reilly Media, 2016.

[30] Flycap application. `https://www.flir.com/products/flycapture-sdk`. Accessed: 2019-03-01.

[31] Camera intrinsic calibration. `http://wiki.ros.org/camera_calibration-intrinsic calibration`. Accessed: 2019-03-08.

[32] Finding lane lines for self driving cars. `https://github.com/rkipp1210/pydata-berlin-2017`. Accessed: 2019-04-16.

[33] Multitarget-tracker. `https://github.com/Smorodov/Multitarget-tracker`. Accessed: 2019-04-07.

[34] Pydata berlin. `https://berlin.pydata.org/`. Accessed: 2019-05-09.

[35] Finding lanes for self-driving cars - pydata berlin jul 2017- ross kippenbrock of yhat. `https://pt.slideshare.net/Yhat/finding-lanes-for-selfdriving-cars-pydata-berlin-jul-2017-ross-kippenbrock-of-yhat`. Accessed: 2019-05-09.

[36] Sobel derivatives. `https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html`. Accessed: 2019-05-09.

[37] Advanced lane detction algorithm. `https://github.com/HsucheChiang/Advanced_Lane_Detection`. Accessed: 2019-05-09.

[38] Lanenet ros node. `https://github.com/AbangLZU/LaneNetRos`. Accessed: 2019-05-20.

[39] Davy Neven, Bert De Brabandere, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Towards end-to-end lane detection: an instance segmentation approach. pages 286–291, 06 2018.

[40] Lesson 3: Data blocks; multi-label classification; segmentation. `https://course.fast.ai/`. Accessed: 2019-05-15.

[41] Image segmentation with camvid. `https://nbviewer.jupyter.org/github/fastai/course-v3/blob/master/nbs/dl1/lesson3-camvid.ipynb`. Accessed: 2019-05-15.

[42] Semantic segmentation datasets for urban driving scenes. `https://autonomous-driving.org/2018/07/15/semantic-segmentation-datasets-for-urban-driving-scenes/`. Accessed: 2019-05-15.

[43] Mapillary vistas dataset. `https://www.mapillary.com/dataset/vistas?pKey=q0GhQpk20wJm1ba1mfwJmw` . Accessed: 2019-05-15.

[44] Pod manager tool podman. `https://podman.io`. Accessed: 2019-04-29.

[45] Jupyter. `https://jupyter.org/`. Accessed: 2019-04-29.

[46] What are linux containers? `https://opensource.com/resources/what-are-linux-containers`. Accessed: 2019-04-29.

[47] Launch files. `http://www.clearpathrobotics.com/assets/guides/ros/Launch%20Files.html`. Accessed: 2019-05-12.

[48] David Silva. *Multisensor Calibration and Data Fusion Using LIDAR and Vision.* Master thesis, University of Aveiro, 2016.

[49] Daniela Rato. *Detection of the Navigable Road Limits by Analysis of the Accumulated Point Cloud Density.* Master thesis, University of Aveiro, 2019.