DISSERTAÇÃO DE MESTRADO

Multi-Camera Framework for Object Detection and Distance Estimation

Bruno Justino Garcia Praciano

Brasília, 09 de outubro de 2020

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

DISSERTAÇÃO DE MESTRADO

Multi-Camera Framework for Object Detection and Distance Estimation

Bruno Justino Garcia Praciano

Dissertação de Mestrado submetido ao Departamento de Engenharia Mecânica como requisito parcial para obtenção do grau de Mestre em Sistemas Mecatrônicos

Banca Examinadora

Prof. João Paulo C. L. da Costa, Prof. Dr.-Ing., ENE/UnB, Hochschule Hamm-Lippstadt
Orientador

Rafael T. de Sousa Jr., Prof. Dr., ENE/UnB
Examinador interno

Arnaldo Arancíbia, Dr.-Ing., EFS GmbH, TU Berlin
Examinador externo

FICHA CATALOGRÁFICA

PRACIANO, BRUNO JUSTINO GARCIA
Multi-Camera Framework for Object Detection and Distance Estimation [Distrito Federal] 2020.
xvi, 86 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2020).
Dissertação de Mestrado - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Autonomous Vehicles	2. Computational Vision
3. Object Detection	4. Distance Estimation
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

PRACIANO, B. J. G. (2020). *Multi-Camera Framework for Object Detection and Distance Estimation* . Dissertação de Mestrado, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 86 p.

CESSÃO DE DIREITOS

AUTOR: Bruno Justino Garcia Praciano
TÍTULO: Multi-Camera Framework for Object Detection and Distance Estimation .
GRAU: Mestre em Sistemas Mecatrônicos ANO: 2020

_____

Bruno Justino Garcia Praciano

Dedicatória

Bruno Justino Garcia Praciano

ACKNOWLEDGMENTS

First of all, I would like to thank God for giving me health and concentration during the COVID19 crisis in Europe and not allowing anything wrong to happen to me.

I am eternally thankful to my family Alessandra Justino, Flávio Praciano, Lenita Justino, and Victor Hugo, whom I can count and trust unconditionally.

An invaluable character is my master tutor Prof. Dr.-Ing. Joao Paulo C. L. da Costa, who directed me to the new field of autonomous vehicles, always motivated me to do my best.

I want to express my gratitude to my EFS advisor Lothar Weichenberger for trust in my work and always support me and a special thanks for the company Elektronische Fahrwerksystem GmbH for financial and structural support. Other very essential supporters in my thesis are Lukac Branimir, Tobias Behn, and Andreas Schustek, without their valuable support, and it would not be possible to do much of my work.

Along this road, I had some problems with German bureaucracy, my sincere thanks for Lukas Bös for his unbelievable capacity to solve problems, and his communication skills. Also, I need to thank Sara Martin and Robin Käsmayr for their support and also students who attended our Stammtisch. Moreover, I would like to thank Vanessa Voll for all the help provided.

My time in Germany was excellent due to the great hospitality of my host family. Therefore, I thank Johanna Hirschmann and Herbert Hirschmann for making everything easy.

Additionally, the unconditional help and psychological support of several colleagues were essential. In this sense, I would like to thank my flatmate Gabriel Pinheiro, Lucas Maciel, Yan Trindade, Fábio Mendonça, Daniel Alves, Francisco Lopes, Robson de Albuquerque, Giovanni Almeida, and Iure Brandão.

I also express my sincere gratitude for the professors from the University of Brasilia, particularly for Prof. Dr. Rafael Timoteo de Sousa Junior, for his invaluable support and guidance along with my career Prof. Dr. Edna Canedo, Prof. Dr. Georges Amvame for partnership in some papers publications.

My affectionate and sincere thanks go to Anna Ploner, whose support, care and partnership allowed me to improve my life while I was in Germany.

My sincere gratitude for the Institutional Security Office of the Presidency of the Republic of Brazil (Grant 002/2017) and to FAPDF (Projetos UIoT 0193.001366/2016) for the scholarship and the financial support to conference fees.

ABSTRACT

Autonomous Vehicles can reduce the number of car crashes and the number of fatal victims. Following the German Statistical Department, just in 2019, there were over 2 million car accidents, and more than 90 percent of crashes are caused by human errors (National Highway Traffic Safety Administration, 2015). This work proposes a multi-camera system for object detection and distance measurement using computer vision that it will support some autonomous vehicle tests and improve safety during the tests. Three approaches are performed and compared with the real distance, and just the best technique was included in the proposed framework. In most cases, this error is directly related to meteorological factors and weak communication signals between cameras and the control hardware. The results obtained show that object detection methods guarantee precision with an accuracy above 93 % in ideal conditions and controlled environments. However, accuracy is reduced when obstacles are present in front of the detected object. Additional techniques are also proposed to optimize the positioning of the cameras and the angle of inclination.

RESUMO

Os veículos autônomos podem reduzir o número de acidentes automobilísticos e o número de vítimas fatais. Segundo o Departamento de Estatística da Alemanha, apenas em 2019, ocorreram mais de 2 milhões de acidentes de carro. Este trabalho propõe um sistema multicâmera para detecção de objetos e medição de distâncias por visão computacional que irá apoiar alguns testes de veículos autônomos e melhorar a segurança durante os testes. Três abordagens são realizadas e comparadas com a distância real, e apenas a melhor técnica foi incluída no framework proposto. Na maioria dos casos, esse erro está diretamente relacionado a fatores meteorológicos e sinais de comunicação fracos entre as câmeras e o hardware de controle. Os resultados obtidos mostram que os métodos de detecção de objetos garantem precisão com exatidão acima de 93 % em condições ideais e ambientes controlados. No entanto, a precisão é reduzida quando os obstáculos estão presentes na frente do objeto detectado. Técnicas adicionais também são propostas para otimizar o posicionamento das câmeras e o ângulo de inclinação.

SUMMARY

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AV	Autonomous Vehicles
ML	Machine Learning
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
WIFI	Wireless Network Protocol
IDE	Integrated Development Environment
IP	Internet Protocol Version 4
JSON	JavaScript Object Notation
REST	Representational State Transfer
SQL	Structured Query Language
TCP	Transmission Control Protocol
API	Application Programming Interface
WSN	Wireless Sensor Networks
GPS	Global Positioning
LIDAR	Light Detection and Ranging
RADAR	RAdio Detection And Ranging
VM	Virtual Machine
IEEE	Institute of Electrical and Electronics Engineers

1 INTRODUCTION

Autonomous vehicles (AV) can significantly improve road safety and reduce accidents, as most accidents are caused due to human errors. The market penetration rate of AVs is estimated to be between 24% and 87% by 2045 [1]. According to the German Statistical Department, the number of car crashes in Germany was over 2 million, just in 2019, and some levels of AV can support to reduce this amount [2], and more than 90 percent of crashes are caused by human errors [3].

It is possible to create high impact applications in this area, establishing interrelations and information flows to detect new extended stimuli in scenarios where the infrastructure is physical or mobile [4]. There is also the possibility of applying computer vision (CV) concepts that allow the car to perceive the external environment. Combined with CV tools, the vehicle can interpret the external scenario and connect these outputs using Machine Learning (ML) approaches [5].

One significant challenge for AV is to estimate the distance from the surrounding objects. It is possible to perform this task using the vehicle to infrastructure (V2I) communications or other techniques, such as Global Positioning System (GPS) [6]. Therefore, some studies investigate the AV's acceptance to demonstrate that this new technology can reduce car accidents. A research guided by Xu et al. shows that, among the respondents, 42.35 % and 45.28 % expect low risk and lower insurance premiums for autonomous vehicles, respectively [7].

Considering the possibility of using cameras around the road, this thesis proposes an object detection model using a multi-camera approach. We also implement algorithms responsible for detecting and classifying objects and estimating the distance of the object's position.

1.1 MOTIVATION

Autonomous vehicles are an upcoming reality for future years. Following this trend, it is essential to develop new solutions. In particular, vehicle tracking has become a significant necessity. Wider AV adoption should reduce traffic jams and decrease the number of car crashes [8]. There are many self-drive car automation levels until full automation is reached, as shown in Figure 1.1.

The proposal of this work deals with driving test scenarios of AVs. Monitoring test trials using drones is limited to drone's battery life, amounting to 30 to 45 minutes [9]. This work uses infrastructure that can be positioned along the road as a pole-mounted camera that sends the data to a command center.

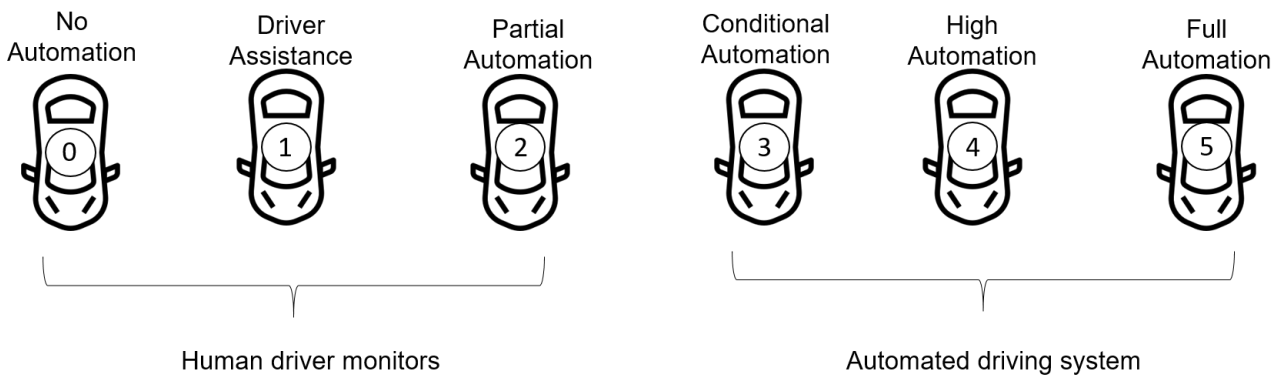

Figure 1.1: Levels of driving automation

1.2 PROBLEM DESCRIPTION

It is to create an architecture to detect objects and to measure their distance along the road. A possible approach is using drones, but it is not ideal due to the battery life limitation for long tests, as illustrated in Figure 1.2. The goal is to track the position of the cars, namely vehicles under the test (VUT) and traffic simulation vehicles (TSV) along the test track, and send this information to the command central (CC).

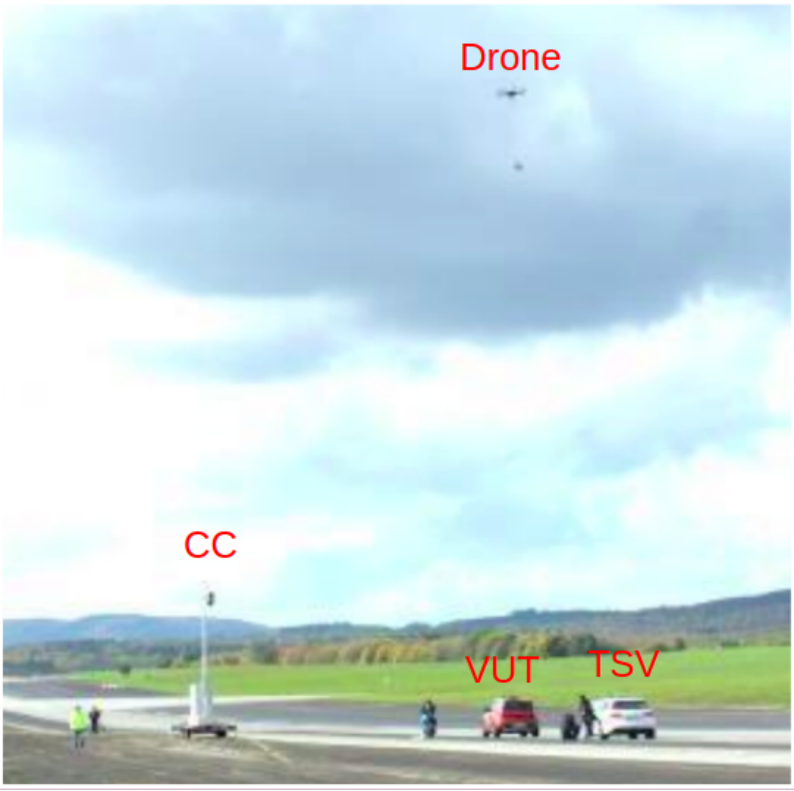

Figure 1.2: Modelling of the test track scenario using drones, CC designates the command central, VUT is the vehicle under test and TSV is a traffic simulation vehicle.

1.3 OBJECTIVES

The main objective is to create a framework to detect objects such as cars, trucks, motorcycles, pedestrians, and other arbitrary obstacles along the road. Additionally, the framework must also estimate the distance of such objects using a camera array.

1.4 PUBLISHED WORKS

Simultaneously, with this work's development, the author has worked in several computer sciences domains, particularly in the data science domain, keeping the multidisciplinary mind. The following works were published along with the pursuit of the Master's Degree:

1. **PRACIANO, BRUNO J. G.**; DE CALDAS FILHO, FRANCISCO L. ; E MARTINS, LUCAS M. C. ; DA CUNHA, DAYANNE F. ; DA SILVA, DANIEL ALVES ; DE SOUSA JÚNIOR, RAFAEL TIMÓTEO. SEGURANÇA DO AMBIENTE USANDO DISPOSITIVO IOT COM PROCESSAMENTO DISTRIBUÍDO. In: Atas da conferência IberoAmericana WWW/Internet 2019, 2019. Atas da conferência Ibero-Americana WWW/Internet 2019, 2019. p. 163

2. MARQUES, Angelica Alves da Cunha; **PRACIANO, Bruno Justino Garcia**. Researchers of the Brazilian archivistics scientific community in French international areas of interlocution Encontros Bibli: revista eletrônica de biblioteconomia e ciência da informação, Florianópolis, v. 25, p. 01-14, mar. 2020. ISSN 1518-2924. doi:https://doi.org/10.5007/1518-2924.2020.e65864.

3. CASTELINO, R. M.; MOREIRA, G. P.; **PRACIANO, BRUNO JUSTINO GARCIA**; SANTOS, GIOVANNI A.; WEICHENBERGER, L.; DE SOUSA, JR, RAFAEL T. Improving the accuracy of pedestrian detection in partially occluded or obstructed scenarios. In: 2020 10th International Conference on Advanced Computer Information Technologies, 2020, Deggendorf. 2020 10th International Conference on Advanced Computer Information Technologies, 2020.

4. CANEDO, EDNA; PINHEIRO, GABRIEL; SOUSA JR., RAFAEL; RIBEIRO, RENATO; **PRACIANO, BRUNO**; LOPES DE MENDONÇA, FÁBIO. Front End Application Security: Proposal for a New Approach. In: 22nd International Conference on Enterprise Information Systems, 2020, Prague. Proceedings of the 22nd International Conference on Enterprise Information Systems, 2020. p. 233.

5. SOUSA JR., RAFAEL; LOPES DE MENDONÇA, FÁBIO; NZE, GEORGES; PINHEIRO, GABRIEL; **PRACIANO, BRUNO** ; CANEDO, EDNA. Performance Evaluation of Software Defined Network Controllers. In: 10th International Conference on Cloud Computing

and Services Science, 2020, Prague. Proceedings of the 10th International Conference on Cloud Computing and Services Science, 2020. p. 363.

6. SILVA, DANIEL ALVES DA; TORRES, JOSÉ ALBERTO SOUSA; PINHEIRO, ALEXANDRE; DE CALDAS FILHO, FRANCISCO L.; MENDONÇA, FABIO L. L.; **PRACIANO, BRUNO J. G**; KFOURI, GUILHERME OLIVEIRA; DE SOUSA, JR, RAFAEL T. Inference of driver behavior using correlated IoT data from the vehicle telemetry and the driver mobile phone. In: 2019 Federated Conference on Computer Science and Information Systems, 2019. org.crossref.xschema._1.Title@7d70270b, 2019. p. 487.

7. KFOURI, GUILHERME DE O. ; GONÇALVES, DANIEL G. V. ; DUTRA, BRUNO V. ; ALENCASTRO, JOÃO F. DE ; FILHO, FRANCISCO L. DE CALDAS ; MARTINS, LUCAS M. C. E ; **PRACIANO, BRUNO J. G.** ; ALBUQUERQUE, ROBSON DE O. ; JR, RAFAEL T. DE SOUSA . Design of a Distributed HIDS for IoT Backbone Components. In: 2019 Federated Conference on Computer Science and Information Systems, 2019. org.crossref.xschema._1.Title@7bad8b1f, 2019. p. 81.

8. DE MENDONCA, FABIO L. L.; DA CUNHA, DAYANNE F.; **PRACIANO, BRUNO J. G.**; DA ROSA ZANATTA, MATEUS; DA COSTA, JOAO PAULO C. L.; DE SOUSA, RAFAEL T. P2PIoT: A Peer-To-Peer Communication Model for the Internet of Things. In: 2019 Workshop on Communication Networks and Power Systems (WCNPS), 2019, Brasilia. 2019 Workshop on Communication Networks and Power Systems (WCNPS), 2019. p. 1.

9. BRANDAO, IURE V.; DA COSTA, JOAO PAULO C. L.; SANTOS, GIOVANNI A.; **PRACIANO, BRUNO J**. G.; JUNIOR, FRANCISCO C. M. D.; DE S. JUNIOR, RAFAEL T. Classification and predictive analysis of educational data to improve the quality of distance learning courses. In: 2019 Workshop on Communication Networks and Power Systems (WCNPS), 2019, Brasilia. 2019 Workshop on Communication Networks and Power Systems (WCNPS), 2019. p. 1.

10. DO NASCIMENTO SILVA, GERSON ; DE CALDAS FILHO, FRANCISCO LOPES ; DOS REIS, VINICIUS ELOY ; **PRACIANO, BRUNO JUSTINO** ; LUSTOSA, JOÃO PAULO ; DE SOUSA JÚNIOR, RAFAEL TIMÓTEO . MODELO DE REDES NEURAIS ARTIFICIAIS EM SUPORTE TECNOLÓGICO À DETECÇÃO DE CARTEIS EM LICITAÇÕES PÚBLICAS. In: Atas da conferência IberoAmericana WWW/Internet 2019, 2019. Atas da conferência Ibero-Americana WWW/Internet 2019. p. 191.

1.5 CHAPTERS DESCRIPTION

This work is structured as follows: in Chapter 2, state of the art is presented to support this work, exploring previous contributions in this research area, the theoretical concepts necessary

to the understanding of this work, and as self-driving cars are a new trend topic, it is essential to describe in greater detail. Chapter 3 shows the proposed framework, and mathematical modeling is presented. Chapter 4 discusses the results achieved, algorithm performance, and the error for object detection and position estimation. Finally, in Chapter 5, the concluding remarks are made regarding the experiments and future works' suggestions.

2 STATE OF THE ART AND TECHNICAL BACKGROUND

This chapter presents a review of the state of the art in Section 2.1. An overview of the main concepts of this work. The present section is divided into three sections. In Section 2.2.1 is defined as the significant concepts of autonomous vehicles. The concepts of sensors are introduced in Section 2.2.2. In Section 2.3, the main ideas regarding machine learning and artificial intelligence are described.

2.1 STATE OF THE ART

This section presents an overview of the related works about this topic. These papers were selected to support the proposed framework during the coding step. These bibliographies support for comparison and the new approach proposal.

The authors from the paper [10] present the next-generation driver assistance systems that require precise self-localization. The system's accuracy is evaluated by computing two independent ego positions of the same trajectory from two different cameras and investigating them for consistency. The landmark creation of the map is one of the significant contributions to the processing pipeline. The rest of the paper assumes a backward-facing stereo camera setup. Moreover, they calibrated all sensors. And the coordinates transform from camera to receiver to be fully known. Its localization presents on a two-step approach and yields a six degrees of freedom ego pose estimation.

The authors of [11] state that the video cameras are among the most commonly used sensors in many applications, ranging from surveillance to smart rooms for video conferencing. In this regard, this paper's primary focus is to highlight the efficient use of the geometric constraints induced by the imaging devices to derive distributed algorithms for target detection, tracking, and recognition. The authors introduced the concept of the image's triangulation, and in many detection and tracking applications, once there are correspondences between object locations across views, the localization of these objects in scene coordinates is most important. Each camera gives rise to a line, and estimating the object's location involves computing the point of intersection of these lines. In the presence of noisy measurements, these lines do not intersect at a single point, and error measures such as the sum of squares are used to obtain a robust estimate of the object's location [12].

In [13] the authors remark that the use of drones has seen a tremendous increase in the last few years, making these devices highly accessible to the public. Besides, computer vision is widely used to autonomously detect drones contrasted to other proposed solutions such as RADAR, acoustics, and RF signal analysis, due to its robustness. The authors aimed to combine a multi-frame deep learning detection method, where the frame coming from the zoomed camera on

the turret is overlaid on the wide-angle static camera's frame. Their equipment's nature can group the proposed approaches in the market and academic papers. However, conventional ones cannot detect small commercial uncrewed aerial vehicles. Also, they are flying at relatively much lower velocities, which decreases the Doppler signature. Based on this information, the authors proposed an approach based on cameras.

The paper [14] shows the development of a three-dimensional coordinate system of objects captured by a sequence of images taken in different views. Object reconstruction is a technique that aims to recover the shape and appearance of items. A novel method to reconstruct occluded objects based on synthetic aperture imaging is presented. The proposed method labels occlusion pixels according to variance and reconstructs the 3D point cloud based on synthetic aperture imaging.

In [15] the authors propose an inter-vehicle distance measurement system for self-driving based on image processing. The proposed method uses two cameras mounted as one stereo camera in the rear-view mirror's hosting vehicle. Extensive experiments have shown the proposed method's high accuracy compared to the previous works from the literature. This method could also be used in several systems of various domains in real-time regardless of the object type. The paper [16] presents a multi-camera vehicle detection system that significantly improves the detection performance under occlusion conditions. They also infer vehicle position on the ground plane by leveraging a multi-view cross-camera context.

In the paper [17], the authors proposed two approaches: estimation of distance using an onboard camera, car position detection, and vehicle position detection is a method of specifying the relative position to the road that can serve as a Lane Departure Warning system. The authors of [18] used a surround multi-camera system to cover the full 360-degree field-of-view around the car. Their pipeline can precisely calibrate multi-camera systems, build sparse 3D maps for visual navigation, visually localize the vehicle for these maps, generate accurate dense graphs, and detect obstacles based on real-time depth map extraction.

Unlike existing methods that use pinhole cameras, the implementation of [19] is based on fisheye cameras, whose large field of view benefits various computer vision applications for self-driving vehicles visual-inertial odometry, visual localization, and object detection. It recovers the depths using multiple image resolutions. At the end of the pipeline, the authors fuse the fisheye depth images into the truncated signed distance function volume to obtain a 3D map.

In [20], the authors show that inter-vehicle distance estimation from an in-car camera based on monocular vision is critical. An improved method for estimating a monocular vision vehicle's distance based on the target vehicle's detection and segmentation is proposed in this paper to address the vehicle attitude angle problem. The angle regression model is used to obtain the attitude angle information of the target vehicle. The dimension estimation network determines the actual dimensions of the target vehicle.

In [21], the multipath and non-line-of-sight effects of GPS receivers decrease the precision of the vehicle's self-localization. More specifically, the lateral error is more severe because of the

blockage of the satellites. The lateral distance between building and vehicle estimated by a stereo camera is compared with a 3D building map to rectify its lateral position. Besides, this paper employs an inertial sensor and GPS receiver to decide the car's longitudinal location.

In [22] introduces CityFlow, a city-scale traffic camera dataset consisting of more than 3 hours of synchronized HD videos from 40 cameras across ten intersections, with the longest distance between two simultaneous cameras being 2.5 km. The authors expected this dataset to catalyze research in this field, propel the state-of-the-art forward, and lead to deployed traffic optimization in the real world.

In [23], the authors measure the distance between the ego-vehicle and the target vehicle using a monocular vision. They also eliminate estimation error by changing the vehicle pose, proposing a distance estimation method based on the car pose information. The proposed technique can reduce the possible failure of distance estimation produced by changing an uncrewed vehicle's inclination angle and roll angle. Furthermore, the pose information could also help evaluate distance if the car is on a slope.

In paper [24], a distance determination technique using an image from the single forward camera is presented. Therefore, automatic brightness adjustment and inverse perspective mapping are applied in the proposed scheme. The experimental results confirm that the proposed technique can detect the object's distance in front of the car, where the error is 7.96%.

The paper [25] simulated experiments to verify the feasibility of the proposed method. Meanwhile, physical experiment results show that this method can effectively reduce the outdoor environment impact and improve the calibration and measurement precision. Furthermore, in [26] presents a novel method of camera parameters calibration for obstacle distance measurement based on monocular vision, and the experiment shows that the proposed method is advantageous.

In [27], the authors performed experiments on the KITTI dataset for accurate 3D object detection. They achieved the same results as state-of-the-art in all related categories while maintaining the performance and accuracy trade-off and still run in real-time.

This paper weighted-mean YOLO to improve object detection's real-time performance by fusing RGB cameras and LIDAR information. It implemented a new system using weighted-mean to construct a robust network and compared it with other algorithms. It shows performance improvement of missed-detection [28]. In paper [29] introduced the Complex-YOLO, a state of the art real-time 3D object detection network on point clouds only. This work describes a system that expands YOLOv2, a fast 2D standard object detector for RGB images, by a specific complex regression strategy to estimate multi-class 3D boxes in Cartesian space.

In [30], the proposed method consists of mapping images to a new coordinate system where perspective effects are removed. The removal of perspective associated effects facilitates road and obstacle detection and also assists in free space estimation. The results show this considerably improves the algorithm's effectiveness and reduces computation time compared with the classical inverse perspective mapping.

In [31], a stereo camera calculated the distance considering the angular distance, the distance between cameras, based on the image's pixel. They proposed a method that measures object distance based on trigonometry. In [32], a new framework for tracking multiple objects is presented. They used fusion techniques to achieve this. They tested it on real-world highway data collected from a massively sensorized testbed capable of capturing full-surround information.

In [33], a new intelligent transport system positioning technique that determines the distance between vehicles via image sensor-based visible light communication was implemented. An original novel method determines the distance between two cars using a low camera resolution.

The object detection, classification, and localization in the real world scenario are studied and discussed by [34]. Furthermore, the suggested approach fuses three different object detection algorithms and classification and uses stereo vision for object localization with the opposition. An algorithm for merging the results of the three object detection methods based on bounding boxes is introduced. The proposed fusion algorithm for bounding boxes improves the detection results and provides an information fusion.

2.2 TECHNICAL BACKGROUND

This section contains the technical background necessary for this thesis understanding. It is divided into three subsections: the concepts behind autonomous vehicles, sensors, and machine learning.

2.2.1 Autonomous Vehicles

This section discusses the importance of the autonomous vehicles domain and its applicability to the society, and the problems occurred as well.

2.2.1.1 What are autonomous vehicles

Modern vehicles now have Advanced Driver Assistance Systems (ADAS) which work at several levels of autonomy, with these levels being outlined by the National Highway Traffic Safety Administration (NHTSA). The levels range from 0, no-automation, to 5, full self-driving automation [35]. An example of an ADAS is a parking system, proposed by [36], that uses sensors to find the best way to maneuver a car into a parking space without driver input. Systems such as these are being used in modern semi-autonomous vehicles as driver aids to hand over work from the driver to the car's systems [37]. As technology progresses, there will be a more and more handover of control from the driver to the vehicle, level Four of automation being the fully-autonomous state that is a main talking point in the automotive industry. The level 5 AV will be able to self-drive anywhere ("full automation"), i.e., no cockpits, drivers are not required to be fit to drive, and even they do not require a driving license (every person in a vehicle is a passenger).

There are many open datasets to allow new people to work with autonomous vehicles and hackathons as well. In the special the KITTI Dataset [38] and NuScenes in [39]. These datasets provided data using GPS, Camera, RADAR, and LIDAR.

2.2.1.2 Challenges in Autonomous Vehicles

The cities are not prepared for autonomous vehicles at the same velocity as the industry. Based on the data provided by [40], only 6% of the biggest cities in the United States are considering creating the necessary infrastructure to work with this new reality.

Putting autonomous vehicles on the street is prolonged due to other problems taking care of the network issues, even 3G, 4G, or 5G, the roads' conservation, and the most critical aspect regarding the legislation.

The study and development of these applications will change many things in society, even in the process sector, until companies work with delivery. Nevertheless, it is indispensable to control the traffic and reduce the accidents, It follows the World Health Organization (WHO), every year, over 1 million people lose their lives in car accidents, and only in Brazil is over 47,000 deaths [41].

2.2.2 Sensors

Figure 2.1 shows the correct place for the principal sensors of an autonomous vehicle, model Audi A8 [42]. The subsections 2.2.2.1, 2.2.2.2, and 2.2.2.3 describe camera, radar, and LiDar respectively.

Autonomous vehicles (AV) would be impossible without sensors: they allow the car to see and sense everything on the road and collect the information needed to drive safely [43]. Furthermore, this information is processed and analyzed to build a path from point A to point B and send the appropriate instructions to the car's controls, such as steering, acceleration, and braking [44].

The technology of sensor manufacturing has shown a significant evolution in the last years, having the appearance of a great diversity of sensors and a general decrease in its cost. Thus, from an economic point of view, it became interesting to replace a high-precision, high-cost sensor with several low-cost and less accurate sensors [45]. The combination of multiple low-cost sensors also implied the development of new methods for merging data from various sensors and specific electronics with high processing capacity capable of using the advantages of these new sensors and minimizing their low precision deficiencies [46].

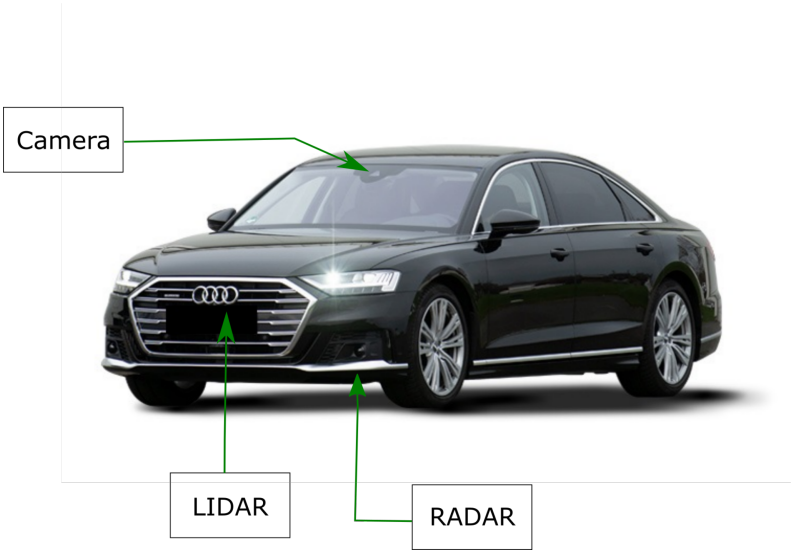

Figure 2.1: Representation of an autonomous vehicle

Each sensor has a different range and coverage angle, and this mixing of the sensor in the car allows it to cover a big area and make it possible to reduce car accidents and improve the driver's safety. Although Figure 2.2 shows reasonable performance parameters for AV sensors, specific sensor designs, and implementations will ultimately determine the performance parameters for a particular AV in the real world.

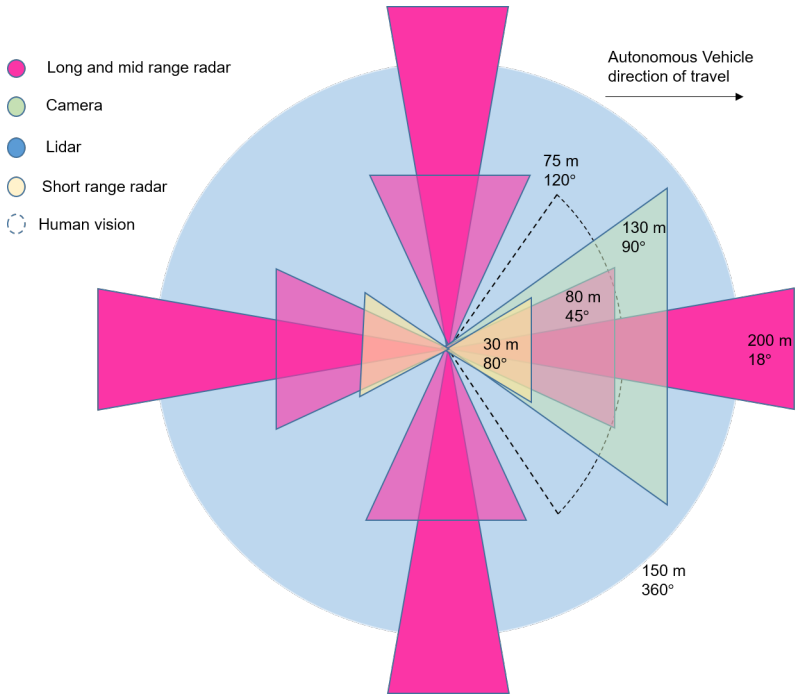

Figure 2.2: Illustration of the various sensors, with reasonable estimates of coverage area (field of view) and typical operating ranges

2.2.2.1 Camera

The camera is the type of sensor that allows the car to perceive the environment using the collected images. Figure 2.4 shows the standard camera for autonomous vehicles.

It is an optical instrument for image or video capture in the same size. For example, there are some cameras where it is possible to set up the numbers of frames per second (FPS) and the image resolution.

It is also essential to remember the importance of the camera's calibration; the Equations for this process were available in [47]. This step is necessary to acquire the objects' real size because it is sometimes required to compare these objects' dimensions.

Figure 2.3: Representation of a camera of the autonomous vehicle [48]

There are many different models of cameras, which vary in their technical specs. In this thesis, it uses GoPro5, which is the highest-end model. The camera alone weighs 118 g and 186 g with the camera. We selected this camera because it works better for outside scenarios. GoPro gives the user great control of the settings for both picture and video acquisition. Video offers some recording options, from 480p until 4k resolution, at various frame rates. There is also the option to shoot in 4K—a resolution higher than most HD televisions. The user can also adjust exposure, white balance, color, ISO, and sharpness, among other settings [49].

GoPro carries a WiFi signal within itself that allows connected devices like another GoPro or your computer. There is no internal storage, and video is saved into removable microSD cards. Battery life is dependent on which video quality. The official Web site suggests a battery life of 2 hours of continuous video and three hours for recharging.

This sensor allows the car to detect many objects while it is possible to perform object recognition. It is necessary to freeze the difference between object detection and object recognition. There are different algorithms to perform these tasks. The proposed algorithm in Chapter 3 shows both exposures combined with the estimation of the distance.

2.2.2.2 Radar

The automotive radio detection and ranging (RADAR) sensors are responsible for performing object detection around the vehicle and avoiding potential collisions. Therefore, with this sensor, it is possible to warn the driver and combined with level 1 of automation, as shown in Figure 1.1, to intervene with the brake the car or use other controls to prevent an accident [50].

Figure 2.4: A commercial radar model ARS430 CV from Continental

Another exciting application of the RADAR in the automotive scenario is measuring the vehicle's relative speed and other objects. It is possible to estimate the distance correctly [51].

It is possible to combine the camera and RADAR and get a new kind of sensor, as defined in [52]. With this approach, it is possible to reach 160 times faster than a human driver.

2.2.2.3 Lidar

In Figure 2.5 is shown a light detection and ranging (LiDAR). The main purpose of this laser is to detect and track any kind of objects.

Figure 2.5: An exemple of a lidar of the autonomous vehicle

This measurement is based on many times rotate this sensor in different directions to scan the whole area of the autonomous vehicles is driving and collect some data for detecting the objects [53].

The name is similar to Radar is not a coincidence. The two technologies exist to understand what is happening in the environment around them, but their methods are different.

The Radar transmits radio waves from a receiver. The radio waves then bounce off objects in the receiver's vicinity to detect how and where said objects are moving. It is available angle and velocity, things of that nature. Think of a weather radar, which measures how quickly and in what direction a storm cloud is moving.

This sensor shoots out invisible beams of light from across the light spectrum. It can use infrared and ultraviolet light to map out the environment around it. It can get a sense of both the physical dimensions and motion (if any) of objects in its vicinity. The usage of the LiDAR is a way to figure out the environment around the object using laser beams.

Autonomous vehicles may use Lidar for object detection and to navigate safely through environments [54]. Point cloud output from the Lidar sensor provides the necessary data for automated software to determine potential obstacles in the ground and where the robot is concerning those potential obstacles. such as CARLA simulator [55] [56].

2.3 MACHINE LEARNING IN COMPUTER VISION

The machine learning is an approach based on algorithms to create some predictions. These techniques are based on mathematical and computer science. It is possible to apply this in several fields of science.

2.3.1 Artificial Neural Networks

The human brain has inspired artificial neural networks (ANN) or perceptron. This approach is because of the capacity of the human mind to categorize new information. In Figure 2.6 is shown an example of the structure of an ANN. There is an input array with the processed features for categorization. The next step of the processing is to define the weights for this analysis. The activation function is the central part of this process. In this step, the algorithm will transform the numbers collected by the previous actions and return only in twofold purpose, like 0 and 1 in the output layer [57].

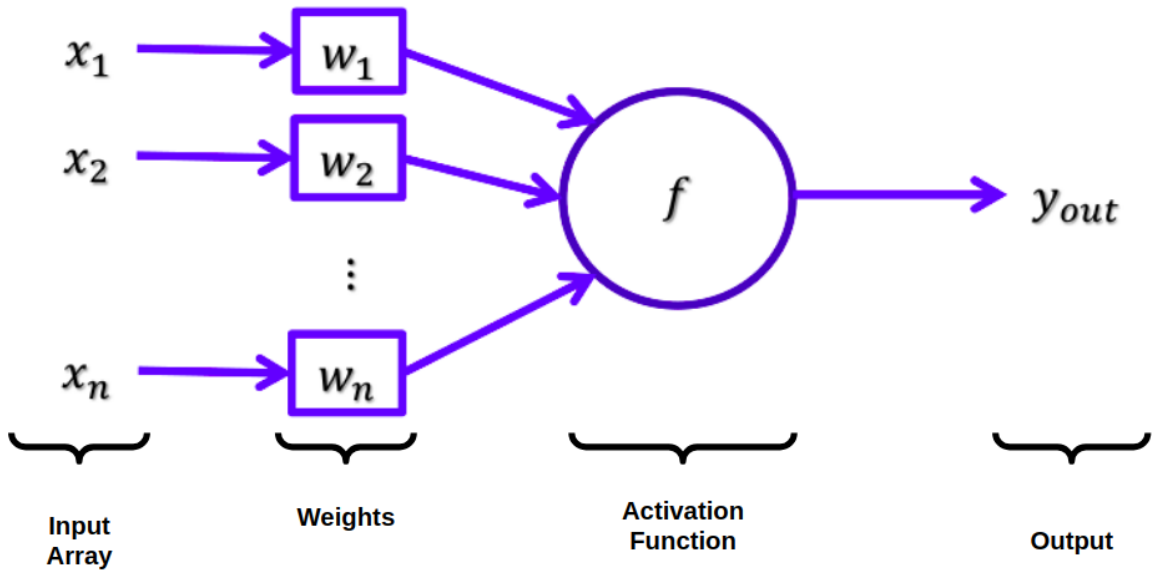

Figure 2.6: The structure of an ANN [58]

The neuron output is a function of the weighted sum of its inputs. Figure 2.7 is shown the mathematical background, where f is activation function, w_i are the weights, and β is the constant input called bias.

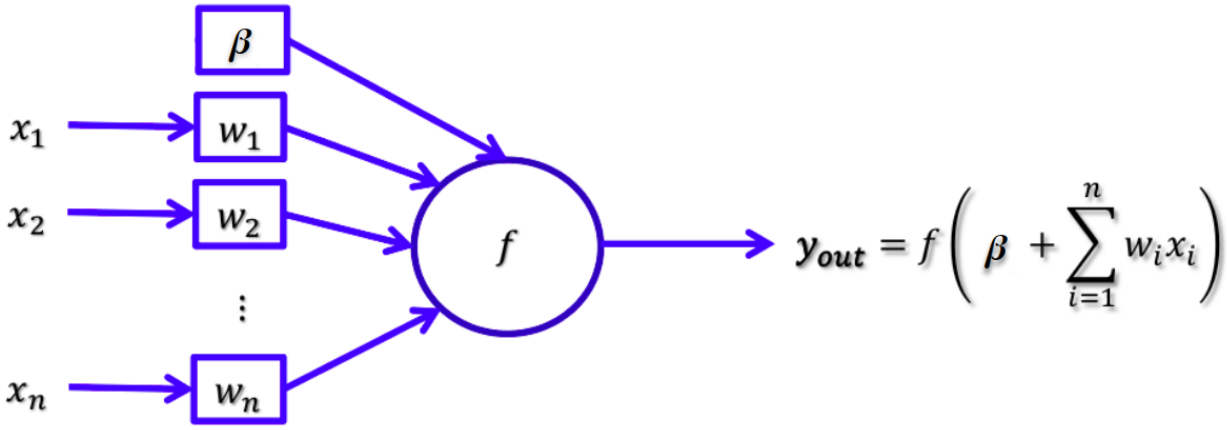

Figure 2.7: Mathematical representation of ANN with bias [58]

2.3.1.1 Activation functions

There are many different activation functions to use. These are crucial for ANN characteristics, such as learning ability and computational efforts in training and validation.

There are different kinds of activation functions for machine learning approaches. Each activation function has its particularity for specific problems. Table 2.1 shows the characteristics for these functions.

Table 2.1: Comparison Table for Activation Functions

Activation Function	**Linear**	**Monotonic**	**Continuous**	**Derivative Monotonic**	**Derivative Continuous**	**Simetric with respect to The Origin**
Unit Step	x	✓	x	x	x	x
Sign	x	✓	x	x	x	✓
Identity	✓	✓	✓	x	✓	✓
Sigmoid	x	✓	✓	x	✓	x
Hyperbolic Tangent	x	✓	✓	x	✓	✓
Rectified Linear Unit (ReLU)	x	✓	✓	✓	x (at 0)	x

Equation 2.1 shows the behavior of the Unit Step activation function ($H(x)$) where it is possible to note that it is nonlinear, monotonic, and adequate for classification. However, on the other hand, it has discontinuous derivatives and not monotonic with gradient descent methods.

$$H(x) = \begin{cases} 1 & if \quad x > 0, \\ \frac{1}{2} & if \quad x = 0, \\ 0 & if \quad x < 0 \end{cases} \tag{2.1}$$

Equation 2.2 shows the behavior of the Sign Function activation function. Its properties are similar to the (2.1) as they have, essentially, the same format with different values. The pros are nonlinear, monotonic, and ideal for fine classification, but the cons are discontinuous and not suitable for regressions.

$$sgn(x) = \begin{cases} 1 & for \quad x > 0, \\ 0 & for \quad x = 0, \\ -1 & for \quad x < 0 \end{cases} \tag{2.2}$$

Equation (2.3) defines the Identity activation function where the usage is indicated when necessary to optimize the weights, but for multilayers networks are not recommended because it is a linear function.

$$f(x) = x \tag{2.3}$$

Equation (2.4) shows the classic activation function for the multi-layer networks. It is important to remind that this function is nonlinear, monotonic, and well-suited to gradient descent, but its asymmetry to the origin may have slower convergence and can get stuck at the training time.

$$\mathrm{S}(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

Equation (2.5) proposes an alternative to (2.4) where has a non-linearity and it is symmetric to its origin, and it is well suited to gradient descent optimization, but at the same, it does not have a monotonic derivative and can have a slow convergence, but it is better sometimes than the Sigmoid activation function.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.5}$$

In this work, ReLU is the activation function chosen activation. Its mathematical definition is shown in (2.6). This activation function is a novel, and it is widely used in deep networks because it is nonlinear and has a fast convergence. Contrarily to the previous equations, it is not differential continuously just at zero, and the issues to work with gradient descent is just around the origin.

$$f(x) = \mathrm{R}(x) = \max(0, x) \tag{2.6}$$

The graphic visualization of the function Relu is shown in Figure 2.8, where this function returns 0 if the number from the weighted sum is lower than 0, or return x if the previous value is over than 0.

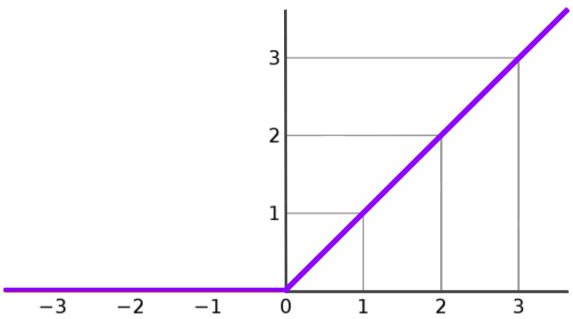

Figure 2.8: The behavior of the Relu [58]

These are the essential characteristics of this activation function. It is widely used in deep networks. It is nonlinear, monotonic, derivative monotonic, and fast converge. As the weaknesses, this activation function has a non continuously differential at zero, i.e., issues with gradient descent around the origin.

2.3.1.2 Layered Neural Networks

The quintessential element of a deep learning model is the multilayer perceptron (MLP) [57]. An MLP is just a mathematical function mapping some set of input values to output values. The function is formed by composing many more specific functions [57].

These are important deep learning models. The goal of a feedforward network is to approximate some function $f*$. For example, for a classifier, $y = f*(x)$ maps an input x to a category

y. A feedforward network defines a mapping $y = f(x; \theta)$ and learns the value of the parameters θ that result in the best function approximation.

These models are called feedforward because information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y. There are no feedback connections in which results of the model are fed back into itself.

2.3.2 Convolutional Neural Networks

The Convolutional Neural Networks (CNN) is a specialized neural network for processing data known as in [59]. For example, in the autonomous vehicle domain, this approach is several used for object detection and object identification. This name indicates that the network employs a mathematical operation called convolution.

A CNN coarsely scans the image for features (in lower dimension space), pools possible patterns, then inspect those patterns in detail with its fully connected subnetworks, generating their classifications. In Figure 2.9 is defined the full process of this neural network. Where there are three other importants is steps: Convolutional layer is defined in Subsection 2.3.2.1. The pooling layer is introduced in the Subsection 2.3.2.2.

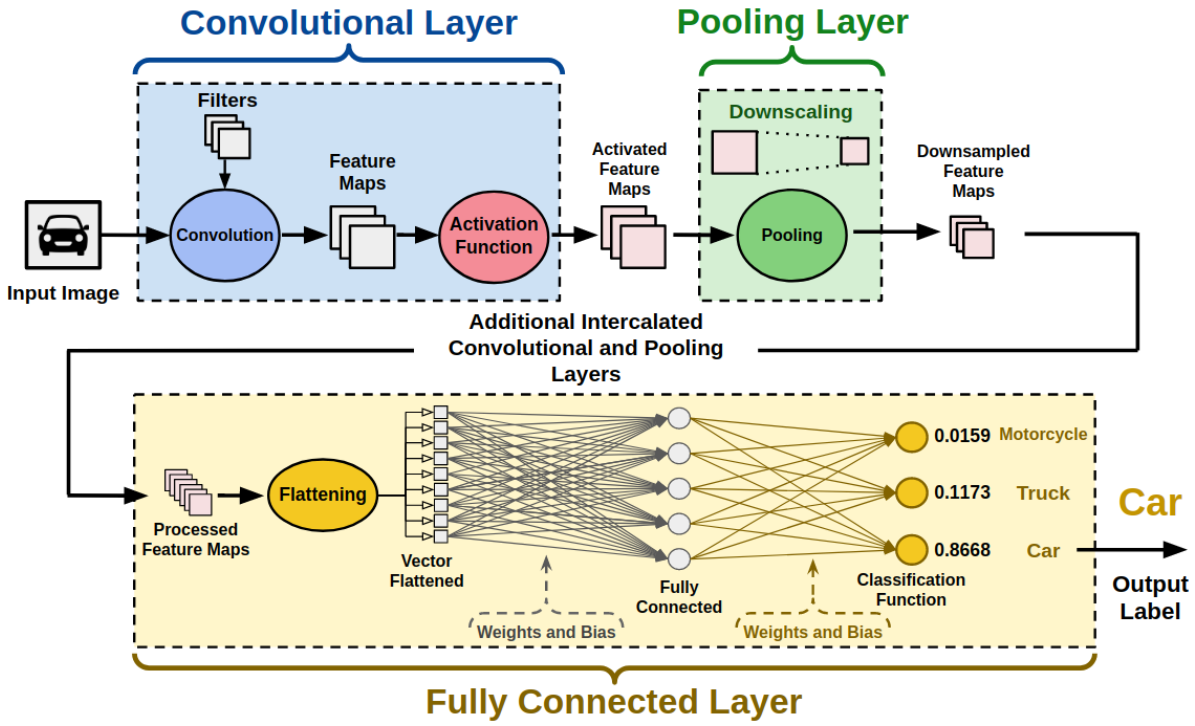

Figure 2.9: Full process of a convolutional neural network [58]

2.3.2.1 Convolutional Layer

The standard inputs are a tridimensional matrix with height and width defined accordingly with the image dimensions and determined by the number of colors. In general, the images use three color channels, Red-Green-Blue (RGB), as is shown in Figure 2.10.

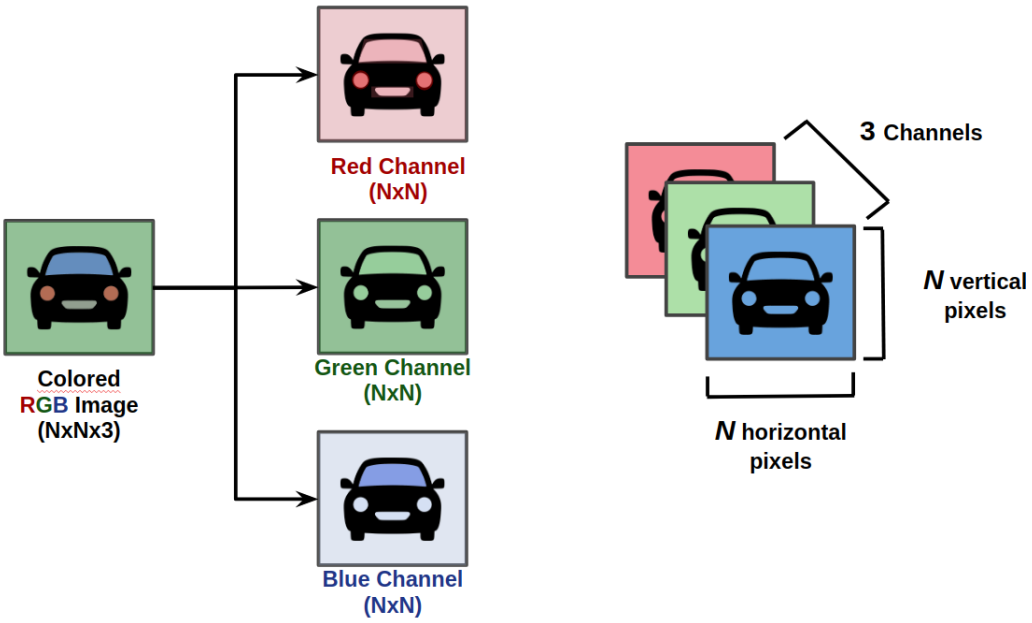

Figure 2.10: Representation of the colors of the input image [58]

The convolutions work as filters that seem little squares, and they are slipping through the whole image and capturing essential parts. Figure 2.11 shows an image by dimensions of $NxNX3$, filter among the $MxMX3$, wherever individual main difference per result is then summed, on bias (β) value, then passed through an activation function. Furthermore, the end of the process generates a new matrix called a feature map or activation map.

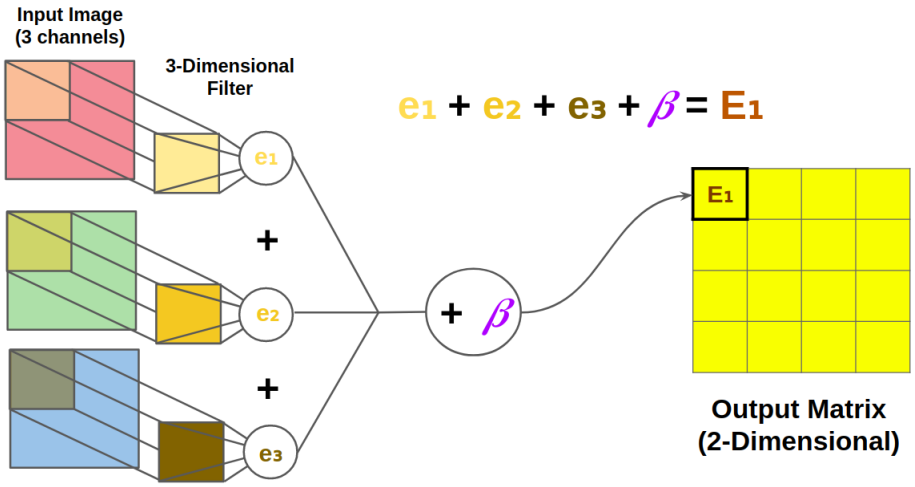

Figure 2.11: Representation of the convolution process [58]

2.3.2.2 Pooling and Upsampling

A pooling layer is necessary to simplify the information from the previous layer. The convolution layer chooses a unit area, for example, $2x2$, to slicing for the whole output information from the previous step. To brief, if the information from the previous layer was $4x4$, the output from the process of pooling will be $2x2$. Nevertheless, the most used method is max-pooling, where the biggest number in the matrix is passed to the next step. This data summarization is used to reduce the number of weights and avoid overfitting. In Figure 2.12 is shown the max-pooling process.

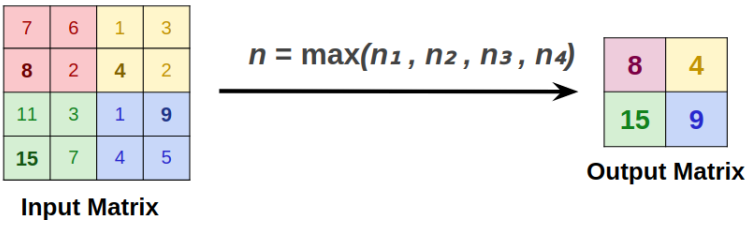

Figure 2.12: Representation of the maxpooling process [58]

2.3.2.3 Auto-encoders

It is a special type of neural network that is used to copy its inputs to its output. The intern structure is defined in Figure 2.13.

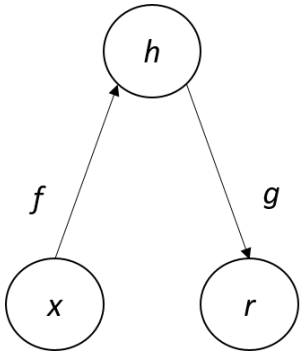

Figure 2.13: The structure of a standard autoenconder, where the variable x means input and r as an output through the internal representation in h. The encoder f maps x to h and decoder g maps h to r

As described in [60], this architecture has a hidden layer of h that describes a code used to represent the input. The network may be viewed as consisting of two parts: an encoder function $h = f(x)$ and a decoder that produces a reconstruction $r = g(h)$.

2.3.2.4 Training

In the machine learning scenario, in special, the neural networks domain epoch can be defined as a single forward pass and backward pass of all the training examples. It feeds in all the

neurons into the network at once. Instead, it chooses a batch of neurons and feeds them in. It performs stochastic gradient descent and prevents the system from overfitting. There is a difference between individual training step time and total training time [61].

3 PROPOSED FRAMEWORK

The framework was idealized to work and different scenarios. In Section 3.1, the proposal with one camera with the object calibration is presented. Section 3.2 defines the problem with one camera but using a known map along metrics to estimate the position along with the map. The Section 3.3 is defined the approach with multi-cameras and real-time processing.

3.1 APPROACH 1 - ONE CAMERA WITH OBJECT CALIBRATION

The first proposed technique is based on the authors of the paper [62], where it is necessary to calibrate the camera before starting the object recognition and classification. This proposal was defined in six steps, as shown in Figure 3.1.

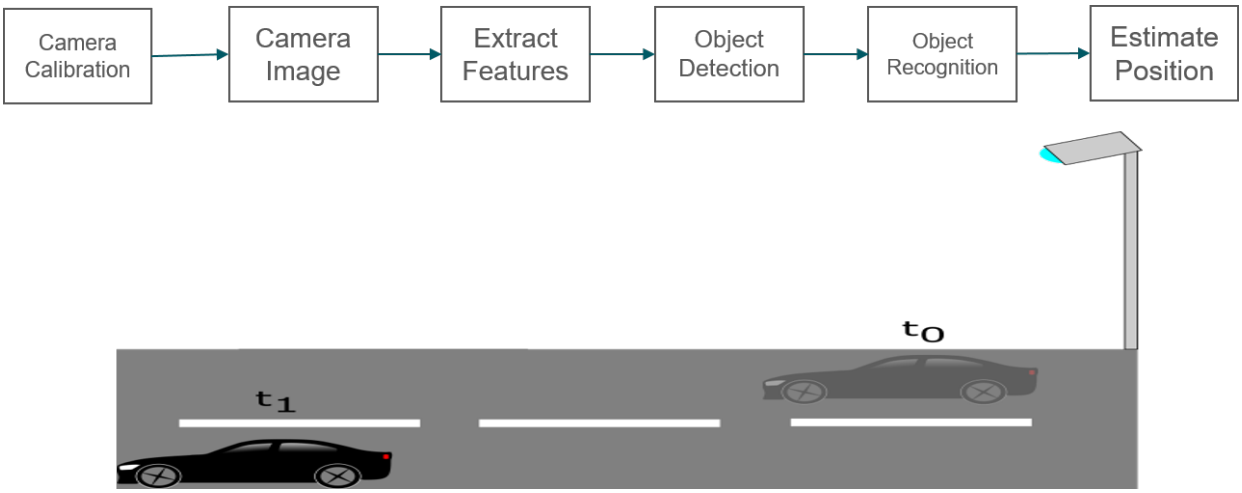

Figure 3.1: Proposal using only one camera with object calibration

3.1.1 Camera Calibration

This task is necessary to reduce the distortion of the camera. The camera used on the tasks has a noise, for this approach is recommended to perform this step. Following this requirement, a script in Python language with the OpenCV library based in [63] was developed. Furthermore, with calibration, also it is possible to determine the relationship between the camera's natural units (pixels) and the real-world units (for example, millimeters).

Using the intrinsic parameters of the camera, as in (3.1), and one point is projected on the image plane.

$$\begin{bmatrix} u' \\ v' \\ z' \end{bmatrix} = P \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{3.1}$$

The 3D point (X_w, Y_w, Z_w) in the world coordinates to its projection (u, v) in the image coordinates. The algorithm's calibration calculates the camera matrix using the extrinsic and intrinsic parameters. The extrinsic parameters represent a rigid transformation from the 3-D world coordinate system to the 2-D camera's coordinate system. These parameters define a projective transformation from the 2-D camera's coordinates into the 2-D image coordinates. Figure 3.2 shows the block diagram where explains the problem to convert a pixel to the real world.

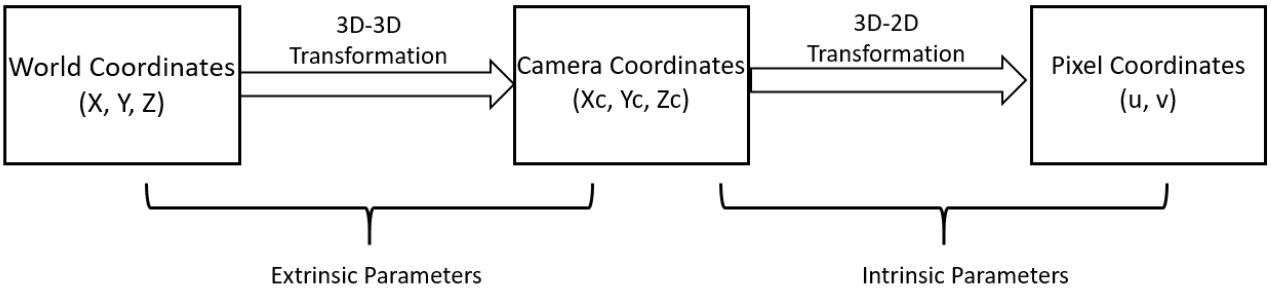

Figure 3.2: Block diagrams of a projection

In (3.2), $\mathbf{P}$ is a 3x4 projection matrix combined of two different parts, the intrinsic parameters of the camera ($\mathbf{K}$) and the extrinsic matrix ($[\mathbf{R}|t]$) that is based on the combination of 3x3 rotation matrix $\mathbf{R}$ and 3x1 translation t vector [64].

$$P = \overbrace{\boldsymbol{K}}^{\text{Intrinsic Matrix}} \cdot \overbrace{[\boldsymbol{R}|\text{t}]}^{\text{Extrinsic Matrix}} \tag{3.2}$$

The intrinsic matrix ($\mathbf{K}$) is an upper triangular matrix as shown in (3.3).

$$\mathbf{K} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

where, f_x, f_y are the x and y focal lengths, c_x, c_y are the x and y coordinates of the center in the image plane, γ is the skew between the axes, in this master's thesis, was defined equal to 0.

The Extrinsic Matrix is shown in (3.4). The extrinsic matrix takes a rigid transformation matrix: a 3x3 rotation matrix in the left-block and a 3x1 translation column-vector in the right. The camera's extrinsic matrix describes the camera's location in the world and what direction it is pointing.

$$[R\,|\,\boldsymbol{t}] = \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{array}\right] \tag{3.4}$$

3.1.2 Camera Image

The image camera model depicted in Figure 3.3 describes the mathematical relationship between the coordinates of a point in 3-dimension space and its projection onto the image plane of a camera aperture is described as a point, and no lenses are used to focus light. The model can only be used as a first-order approximation of the mapping from a 3D scene to a 2D image [65].

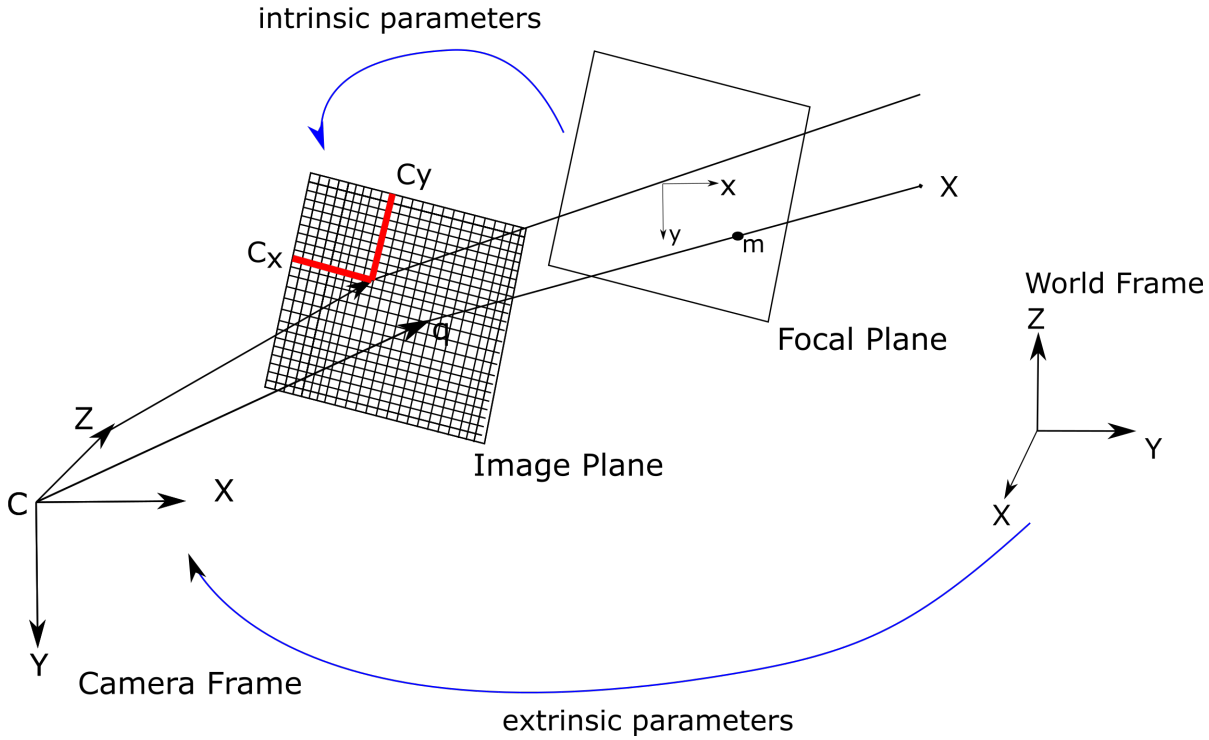

Figure 3.3: The camera model for image formation based on some metrics and known parameters

3.1.3 Features Extraction

When it is necessary to work with variables that contain many contents, there is a necessity to improve this work and reduce the computer bottleneck during the process. In machine learning (ML), some variables are independents or some features on which the final output is done. Moreover, in other cases, that number of these features increases it and reduces the ability to visualize

it.

For example, the image resolution of the collected data for the ML algorithm's training is 1392 pixels in height and 512 pixels in width, for a total of $712,704$ pixels in total. Each pixel has a single pixel-value associated with it, indicating the darkness or lightness of that pixel. The numbers are between 0 and 255, along this premise is necessary to determine which objects the image contains.

The task was performed by feature extraction, which creates new features from existing features, giving us more information and fewer redundancies [66].

A mathematical tool called Principal Component Analysis (PCA) was used in this step, and the PCA is used to decompose a multivariate dataset in a set of successive orthogonal components that explain a maximum amount of the variance [67]. Figure 3.4 is shown how the technique works. The data is decomposed into a perpendicular vector where the information is unrolled. Besides, with more variance means more information regarding data.

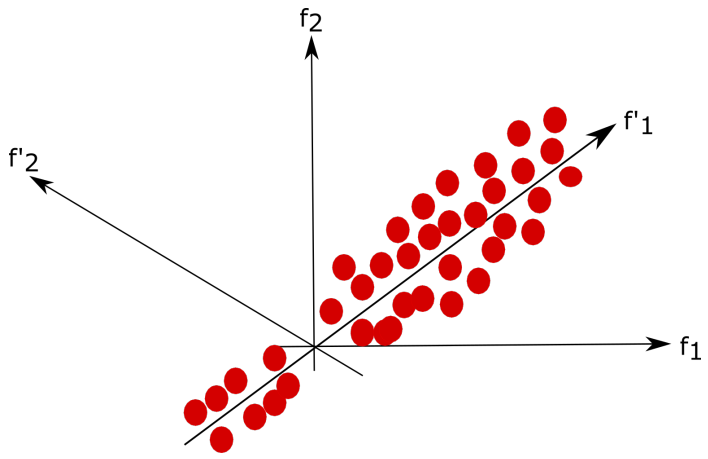

Figure 3.4: Maximum variance in f_1', where the red circles mean the data points of the data set, f_1 is the feature 1 on x-axis, f_2 is the feature 2 on y-axis

Based on Figure 3.5, is necessary to find a direction f_i such as the variance of $x_i's$ project on $f_i's$ has the maximum value. Also, it is necessary to rotate the previous axis to find $f_i's$, and finally, drop f_2

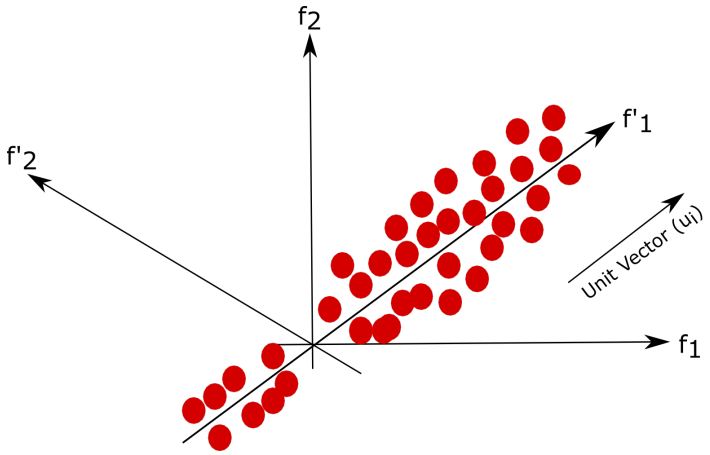

Figure 3.5: Unit vector direction of maximum variance

To find the direction of $f_i's$, which has the maximum variance, unit vector in the direction of maximum variance = U_i, in (3.5a) is described how to compute this distance.

$$x_i' = \text{Projection of } x_i \text{ on unit vector } u_i \tag{3.5a}$$

$$= u_i^T x_i \tag{3.5b}$$

$$\overline{x_i'} = u_i^T \cdot \underbrace{\overline{x}_i}_{Mean\ Vector} \tag{3.5c}$$

$$var\left\{u^T x_i\right\}_{i=1}^{n} = \frac{1}{n}\sum_{i=1}^{n}\left(u_i^T x_i - \underbrace{u_i^T \overline{x}_i}_{Mean\ \overline{x}_i}\right)^2 \tag{3.5d}$$

In (3.5d) is possible to find out the u_i which gives the maximum variance. Further, this problem can be defined as distance minimization [68].

In Figure 3.6 the vector which gives the minimum distance $(d_1, d_1, \cdots)$ when $x_i's$ are projected on u_i.

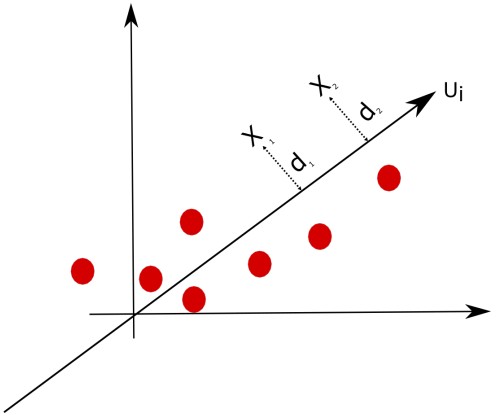

Figure 3.6: Distance minimization PCA

In (3.6), the equation finds the vector u_i, which gives the minimum distance.

$$d_i^2 = \|x_i\|^2 - \left(u^T x_i\right)^2 \tag{3.6a}$$

$$= \left(x^T x_i\right) - \left(u^T x_i\right)^2 \tag{3.6b}$$

$$min_{u_i} \sum_{i=1}^{n} \left(x_i^T x_i - \left(u^T x_i\right)^2\right) \tag{3.6c}$$

Calculation of Eigenvalues and Eigenvectors give the solution to the above Equations.

where the matrix $\mathbf{X}$ in (3.7) is the matrix of the data points with the shape (nxd)

$$\mathbf{X} = \begin{bmatrix} a_{11} & a_{12} & \dots \\ \vdots & \ddots & \\ a_{K1} & & a_{KK} \end{bmatrix} \tag{3.7}$$

The square symmetric matrix is defined as $S_{dxd} = X_{dxn}^T X_{nxd}$ [69].

Based on the approach of [70], in (3.8) is defined the solution equation.

$$\lambda_i V_{i_{dx1}} = S_{dxd} V_{i_{dx1}} \tag{3.8}$$

where λ is the scalar eigenvalues, S is the co-variance matrix, V is the vector - eigenvector, and d is the dimension.

The steps to find the Eigenvector:

1. Do the column standardization of $\mathbf{X}$
2. compute the co-variance Matrix: $S = X^T X$
3. $\lambda =$ Eigen Value and V = Eigen Vector
4. $\lambda V = SV$

To brief these steps is necessary to assume the more variability in a particular direction correlates with explaining the dependent variable's behavior. Theoretically, it is needed to apply the PCA to remove the sample's noise and keep only the necessary things to detect.

3.1.4 Object Detection and Object Recognition

This task is based on the paper [71], where You Only Look Once (YOLO) version 3 is used as an object detector and uses the features after the pre-processing as input the deep convolutional neural network in Figure 3.7.

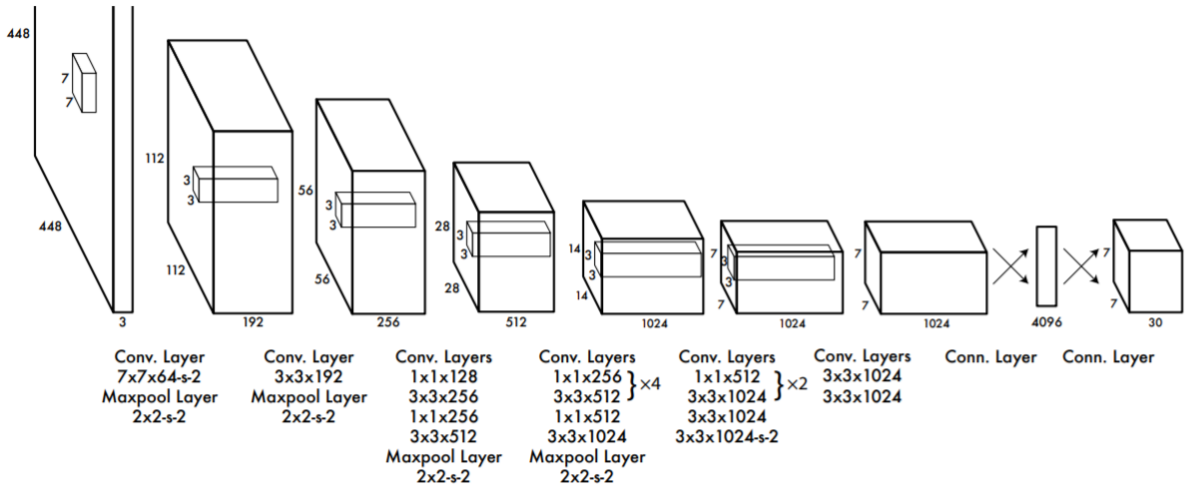

Figure 3.7: YOLO Architecture: Simultaneously predicts bounding boxes and class probabilities for these boxes [71]

This architecture makes use of only convolutional neural networks. This topic has already been detailed in Subsection 2.3.2 and makes it in a fully convolutional network (FCN). Yolo has 75 convolutional layers, with skip connections and upsampling layers.

In the YOLO environment, the algorithm divides the input image into a ZxZ grid. Each grid of this frame predicts only one object, as shown in Figure 3.8. Along, YOLO uses $7x7$ grids

(ZxZ), two boundary boxes (B), and 20 classes (C). So, the tensor of the YOLO prediction has a shape of $(Z, Z, Bx5 + 20) = (7, 7, 30)$

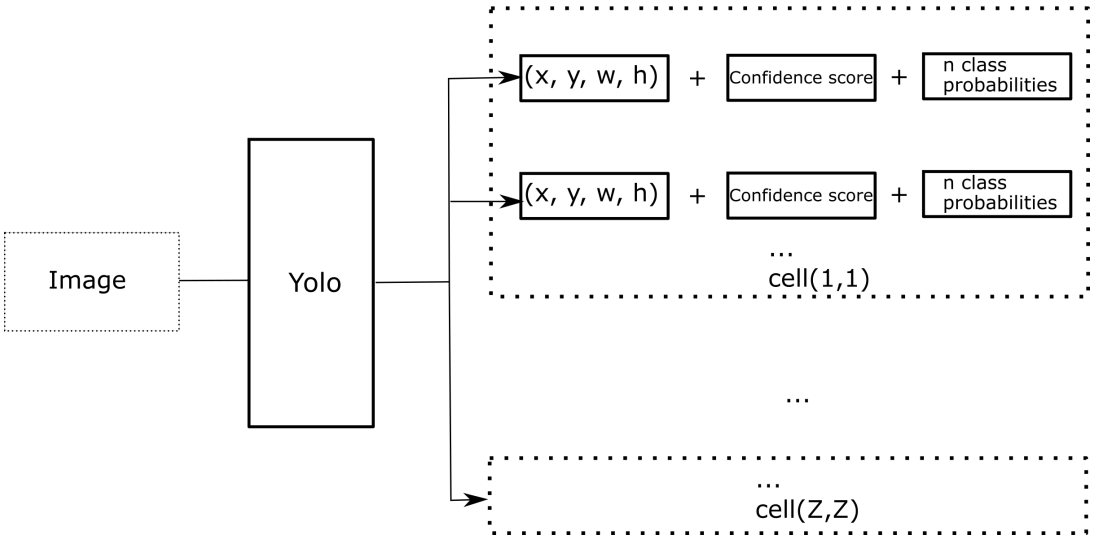

Figure 3.8: Yolo makes ZxZ predictions with B boundaries boxes

The object detection is based on the boundary box approach, and each box has five known elements (x, y, w, h) and a box confidence score. This score means how likely the box contains an object. It uses CNN to reduce the spatial dimension; after that, it performs a linear regression using two fully connected layers to make the predictions; this approach considers only predictions over 0.5. It is defined in Table 3.1.

Table 3.1: The Yolo's predicts equations

Description	Equation
box confidence score	$P_r(object).IoU$
conditional class probability	$P_r(class_i\|object)$
class confidence score	$P_r(class_i).IoU$
class confidence score	box confidence score · conditional class probability

where in Table 3.1,$P_r(object)$ is the probability the box contains an object. IoU is the intersection over the union between the predicted box and the ground truth. $P_r(class_i|object)$ is the probability the object belongs to $class_i$ given an object is presence. $P_r(class_i)$ is the probability the object belongs to $class_i$.

The bounding boxes concept is defined in [72], and in the many problems as in the autonomous driving domain, the most common detection will be pedestrians and cars at different distances [73]. It is necessary to apply the clusterization approach. In this case, it is defined by K-means

with $K = 5$. Since the algorithm is working with many kinds of bounding boxes, it is impossible to use the regular spatial distance to measure the data point distances. That is the reason to use IoU. Based on the length of the cluster called as the anchor, in this solution will predict five parameters (t_x, t_y, t_w, t_h, and t_o) combined with the sigma function to reduce the offset range as is already defined in (3.9) and it is detailed graphically in Figure 3.9.

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \\ P_r(object) \cdot IoU(b, object) &= \sigma(t_o) \end{aligned} \tag{3.9}$$

where t_x, t_y, t_w, t_h are the predictions made by the algorithm. c_x, c_y are the top left corner of the grid cell of the anchor. p_w, p_h are the width and height of the anchor. The image width and height normalize c_x, c_y. b_x, b_y, b_w, b_h are the predicted boundary box. $\sigma(t_i)$ is the box confidence score.

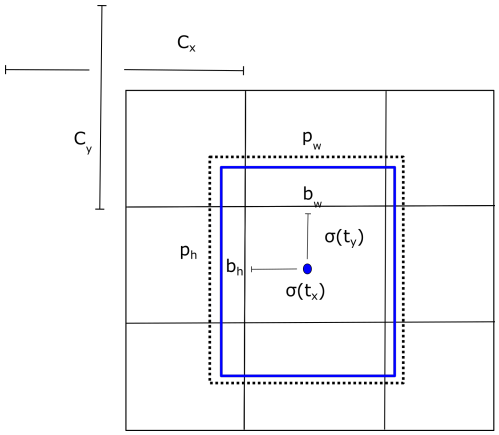

Figure 3.9: Prediction of the width and height of the box as offsets from clusters centroids based on [72]

As already defined, the proposed solution predicts multiple bounding boxes per grid cell. Thus, it is necessary to compute the loss for the true positive, to reduce the error. Hence the object is to be faster, not accurate. On the other hand, each cell will be looked at time and use, and it will be used the highest IoU. The loss function is composed by classification loss in (3.10), the localization loss in (3.11), the confidence loss (3.12), and the loss function is (3.13).

If the object is located on the frame, the classification loss will perform the squared error at each cell on the conditional probability for each class:

$$\sum_{i=0}^{s^2} 1_i^{obj} \sum_{c \in \ classes} \left(p_i\left(c\right) - \hat{p}_i\left(c\right)\right)^2 \tag{3.10}$$

where 1_i^{obj} is the Boolean that controls if has an object or not, $\hat{p}_i\left(c\right)$ denotes the conditional probability for each class in the cell.

The localization loss (loc_{loss}) is necessary to take care of the measurement errors regarding the locations and the boxes' sizes. The goal is not to define the absolute weight errors in large boxes and small boxes. It predicts the square root of the bounding box width and height instead of the width and height.

$$\begin{aligned} loc_{loss} = \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_i^{obj}{}_j \left[(x_i - \hat{x_i})^2 + (y_i - \hat{y_i})^2\right] \\ + \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_i^{obj}{}_j \left[(w_i - \hat{w_i})^2 + (h_i - \hat{h_i})^2\right] \end{aligned} \tag{3.11}$$

where $1_{ij}^{obj} = 1$ if the boundary box in the cell is responsible for detecting the object, otherwise is 0. λ_{coord} increases the weight for the loss in the boundary boxes coordinates, with this variable is possible to put more emphasis on the accuracy, so it is multiplied by the loss, the default value for this work is 5.

The confidence loss ($Conf_{loss}$)is used to measure the box's objectness because a significant part of the boxes does not have any detector inside, and with this, an imbalance issue is noted to avoid this object is necessary to compute this loss.

$$Conf_{loss} = \lambda_{noobj} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{noobj} \left(C_i - \hat{C}_i\right)^2 \tag{3.12}$$

where 1_i^{noobj} is the complement of 1_i^{obj}, $\hat{C}_i$ is the box confidence score of the box j in cell i, and λ_{noobj} takes care of the weights decrease the loss when the background is detected in this work the used value for this variable is 0.5.

The final loss ($loss$) is computed through the addition of previous losses, in (3.13) is defined as the actual loss to reduce the errors in the object detection.

$$
\begin{aligned}
loss = \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_i^{obj}{}_j \left[(x_i - \hat{x_i})^2 + (y_i - \hat{y_i})^2 \right] \\
+ \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_i^{obj}{}_j \left[(w_i - \hat{w_i})^2 + (h_i - \hat{h_i})^2 \right] \\
+ \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{noobj} \left(C_i - \hat{C_i} \right)^2 \\
+ \lambda_{noobj} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{noobj} \left(C_i - \hat{C_i} \right)^2 \\
+ \sum_{i=0}^{s^2} 1_i^{obj} \sum_{c \in \ classes} \left(p_i \left(c \right) - \hat{p}_i \left(c \right) \right)^2
\end{aligned}
\tag{3.13}
$$

After object detection, it is necessary to perform the object classification, where each box predicts the classes the bounding box, so it is recommended to use multilabel classification. The difference in this work is to use the softmax function in the output of the categories. The data used in this training is labeled, and it was collected from Open Image Dataset [74], and this classification was performed over the Darknet neural network [75].

3.1.5 Distance Estimation

This approach uses the object detector's outputs for the distance estimation, where 4 variables are predicted, (x, y, w, h). These work variables x, y are used to adjust the boundary box, and w, h are used in Figure 3.8 to measure the distance of the object. These variables will variate according to the distance of the camera. In [76] the image will be refracted in the lens, and with this is possible to deduce a relationship between the known parameters: focal length (f), the distance of the object from the lens (d), the distance of the refracted image from the lens (D). In Figure 3.10 is shown how the distance measurer works.

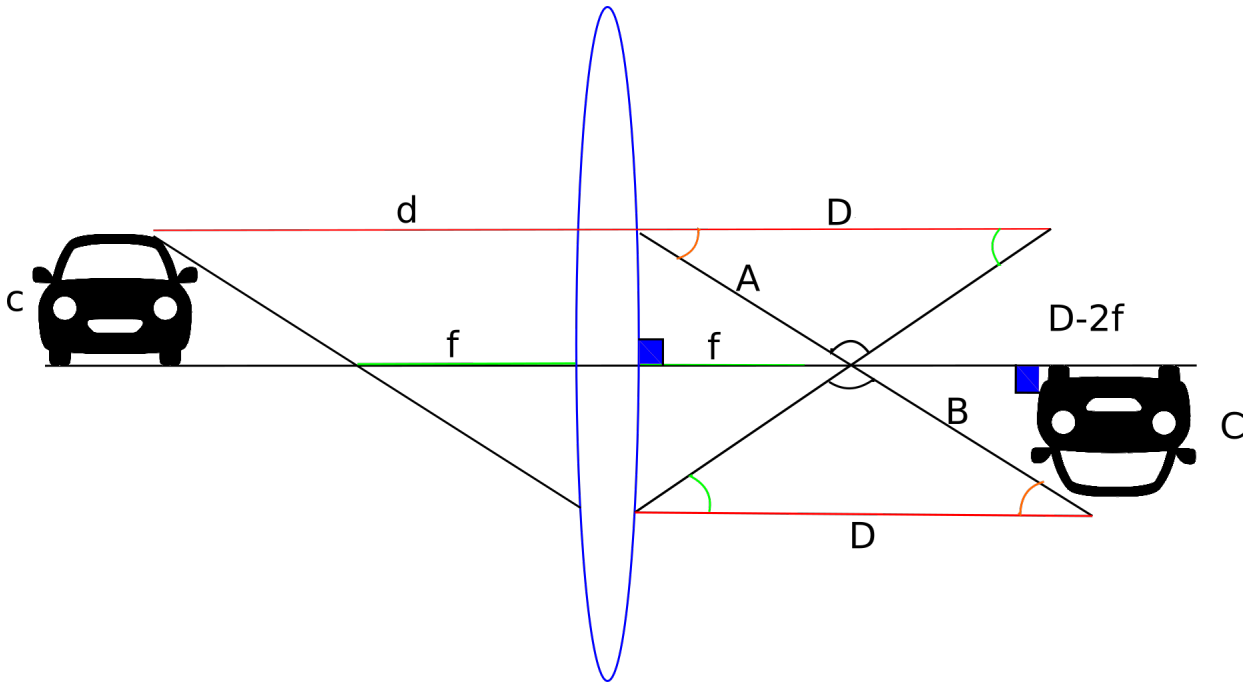

Figure 3.10: Purpose method to compute the distance of the object using cameras

So the red line d represents the actual distance of the object from the convex length. Moreover, D gives a sense of how the actual image looks. If we consider a triangle on the left side of the image (new refracted image) with base d and draw a triangle similar to the left side one. So the new base of the triangle will also be done with the same perpendicular distance. If we compare the two triangles from the right side, we will see d, and D is parallel, and the angle that creates on each side of both the triangle is opposite to each other. From which it is possible to infer that both the triangles on the right side are also similar. Now, as they are similar, the ratio of the corresponding sides will also be similar. So $\frac{d}{D} = \frac{A}{B}$. Again if we compare two triangles on the right side of the image where opposite angles are equal, and one angle of both the triangles are right angle (90°) (dark blue area). So A and B are both hypotenuses of a similar triangle where both triangles have a right angle. So the equation is defined as:

$$\frac{d}{D} = \frac{A}{B} = \frac{f}{D - f}, \tag{3.14}$$

the focal distance is shown in (3.15),

$$\frac{1}{f} = \frac{1}{d} + \frac{1}{D}, \tag{3.15}$$

The proportional size of each image, as shown in (3.16), belong to object detection variables, as shown in (3.18).

$$d = f + \frac{C}{c}, \tag{3.16}$$

the focal length is computed by (3.17),

$$f = \frac{2 \cdot 3.14 \cdot 180}{360}, \tag{3.17}$$

Finally, it is possible to predict the distance based on outputs from the predictor combined with fundamental physics in (3.18), where w is the width and h the height of the object.

$$distance = \frac{2 \cdot 3.14 \cdot 180}{w + h \cdot 360} + \frac{C}{c} \tag{3.18}$$

3.2 APPROACH 2 - ONE CAMERA WITH KNOWN MAP

The second proposed approach is based on [77], where it is necessary to take the photos and label these images [78]. Its output is shown in Table 3.2, and indicates the position and size of the boundary box as already defined in Figure 3.9, and the real position of the car on the actual scenario, and show in Figure 3.11 is shown the block diagrams and the proposed approach to predict the distance based on the known map. The subsection defines only the step regarding the estimated position based on the map because the other actions have already been described in Section 3.1.

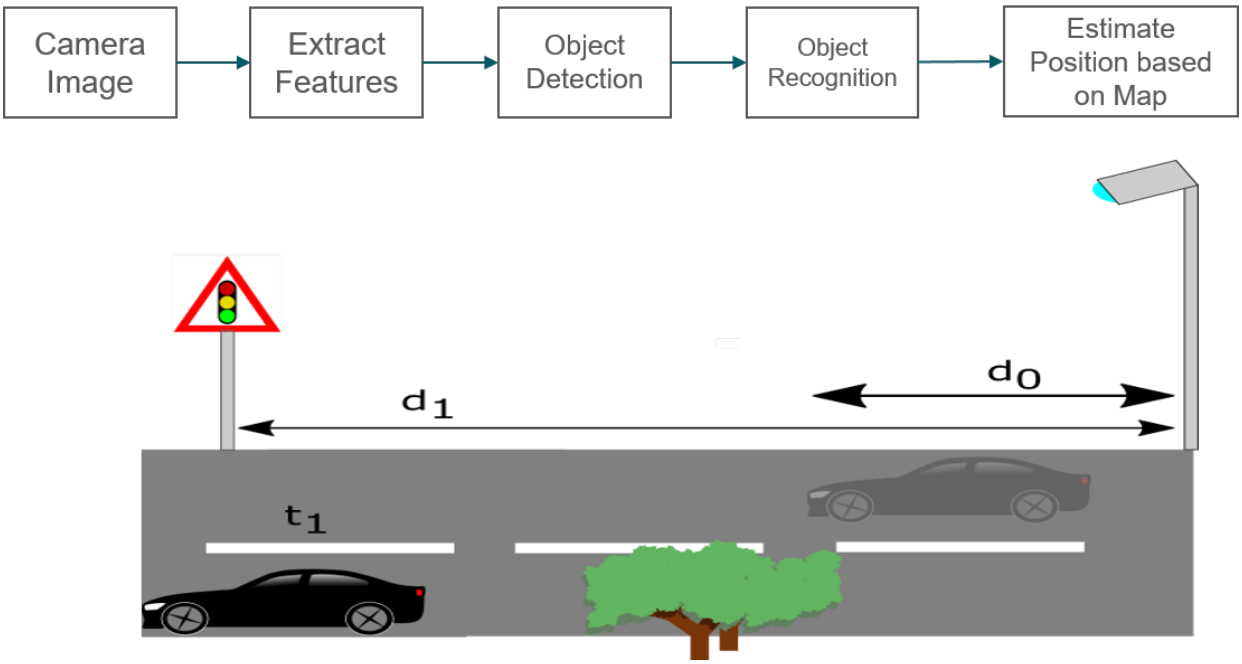

Figure 3.11: Approach using one camera with known map

Table 3.2: Example of labeled values

Label	Distance (meters)	X	Y	W	H
car #1	4.41	365	304	1150	563
car #2	11.11	321	256	736	422
car #3	16.24	221	198	562	351
car #4	19.66	138	172	425	296
car #5	23.09	107	150	360	265
road signal	25.82	1226	6	1266	95
tree	17.22	507	1	606	231

3.2.1 Estimate position based on map

For this step is necessary to collect data and label this data before the start. This predictor will be different from the previous collect data compared with the estimation provided in Section 3.1. This approach is used as an Artificial Neural Network (ANN), and the concepts of this architecture were defined in Subsection 2.3. The proposed ANN is in Figure 3.12.

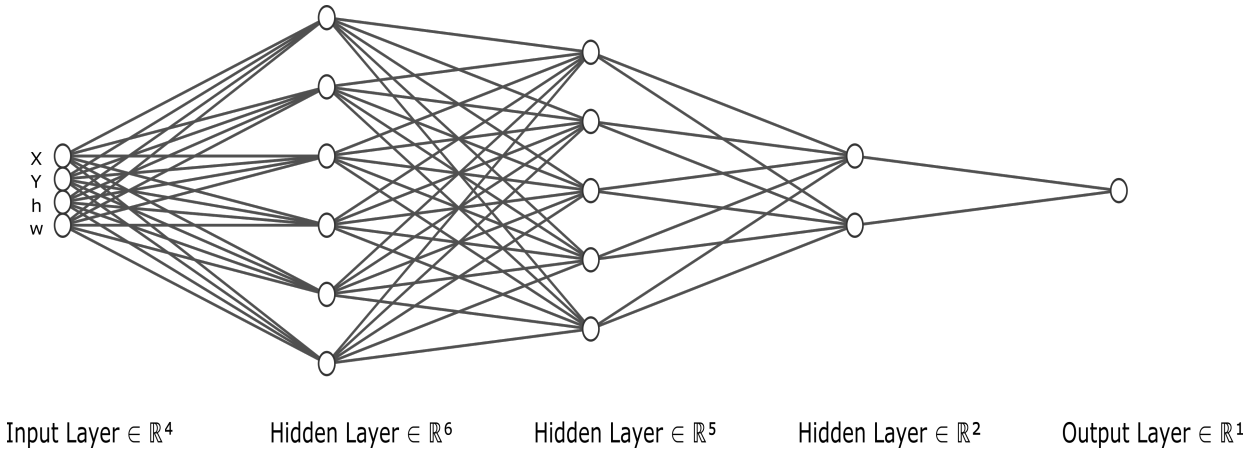

Figure 3.12: Neural network responsible to predict the distance of the objects based on the boundary boxes

Additionally, this defined architecture is possible to estimate the car's distance along with the scenario. An example of the labeled image is shown in Figure 3.13. These outputs were collected from this image and saved in Extensible Markup Language (XML), and this file is used as input in the ANN.

Figure 3.13: Labeled image with boundary boxes positioned in each important element of the screen

3.3 APPROACH 3 - MULTICAMERA

This approach was selected one for constructing the framework in Subsection 3.4 because, with this approach, it is possible to consider the camera's position on the test scenario. As similar in Subsection 3.2, only the last step will be defined here because the other ones were already described in Subsection 3.1.

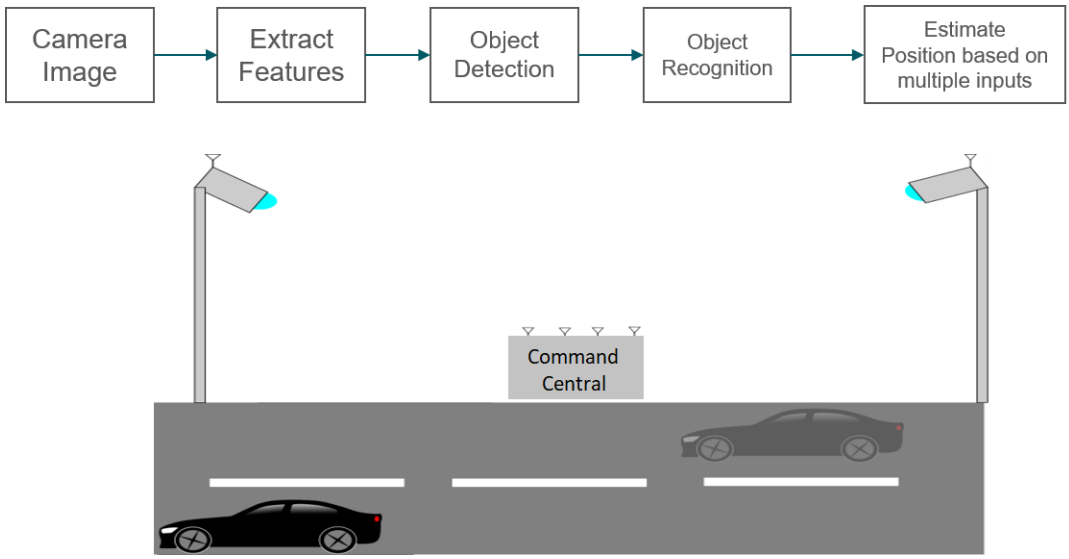

Figure 3.14: Approach using multicamera

The creation of a scenario-based in multiple cameras, forming cameras array, and a command center is responsible for merging all of the collected data and fusing it on the database, such as the label of each object, position, and its timestamp.

The data used on this task is provided from streams of the cameras, and they send these data to the command center via a wireless connection over the protocol IEEE 802.11. The data fusion is controlled by proxy, and it was written in Python. The distance estimation is based on the Inverse perspective mapping (IPM) and is well defined in 3.3.1.

3.3.1 Estimate position based on multiple inputs

Inverse perspective mapping is a mathematical technique that removes the effects of a picture's distortion when transforming the image's perspective to another perspective. Despite disparity mapping, the inverse perspective mapping method requires only one camera, and this method cannot provide depth information directly [79].

The camera must be located in front of the car with an angle of θ to down. Figure 3.15 shows the setup.

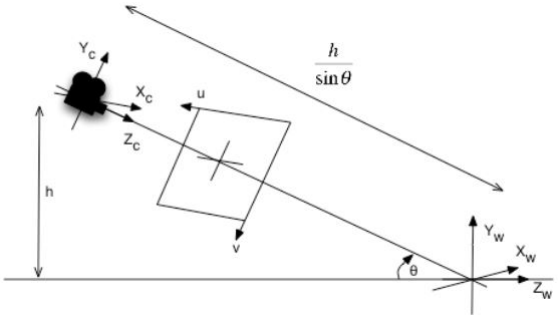

Figure 3.15: Image coordinate system in relation to world coordinate system.

This setup was selected based on solution of [24], the mathematical background is to create top-down view, the surface road point is known as (X_w, Y_w, Z_w) that projects to the image plane (u, v) is a must. As disrupted in Figure 3.15. For rotatation angle (θ), which is angle between camera and the surface, the IPM equation is based on [80] and is shown is Equation 3.19:

$$(u, v, 1)^T = K \cdot T \cdot K(X_w, Y_w, Z_w, 1)^T \tag{3.19}$$

where R is the rotation matrix given in the equation 3.20.

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.20}$$

T is the translation matrix given in the equation 3.21, where h means the height of the position of the camera.

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{-h}{\sin\theta} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.21}$$

K is the camera parameter matrix given in (3.22), where f is the focal length of the camera, s is the skew parameter, and u_0, v_0 are the center of the pixel of desired image size.

$$K = \begin{bmatrix} f & s & u_0 & 0 \\ 0 & f & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.22}$$

The Equation 3.22 can be replaced using the real parameters of this test scenario and these parameters are $f = 2.92mm, s = 0, u_0 = 240, v_0 = 160$. Replacing the Equations 3.20,3.21, 3.22 into the initial Equation 3.19, achieving the new Equation 3.23.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{3.23}$$

where the matrix P was gotten from a product between K, T, and R. As is only necessary to evaluate the position of the road, so the coordinate Y_w can be equal to 0, so simplifying the Equation 3.23, so it is given by Equation 3.24.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{14} \\ P_{21} & P_{22} & P_{24} \\ P_{31} & P_{32} & P_{34} \end{bmatrix} \begin{bmatrix} X_w \\ Z_w \\ 1 \end{bmatrix} \tag{3.24}$$

Based on the Equations above, it is possible to infer the Equation 3.25 to compute the distance from the camera until the object.

1. Calculating average intensity in the row direction from the bottom row up to top row
2. The average intensity of each row is compared with the threshold level (obtained from the experimental), which is 50. The starting position of an indicated object is the average intensity in that row is greater than 50, and the order of that row is stored in a parameter p.
3. The distance between object and vehicle is therefore calculated using a linear equation given in 3.25.

$$d = ap + b \tag{3.25}$$

d is the distance between the camera, object, and the vehicle in meter, p is the order of the row that object is detected, and a, b are constants.

3.4 FRAMEWORK ARCHITECTURE

This section discusses how the architecture of the Subsection 3.3 is encapsulated in a software framework. This Architecture was divided into four big modules: client, model, proxy, and controller.

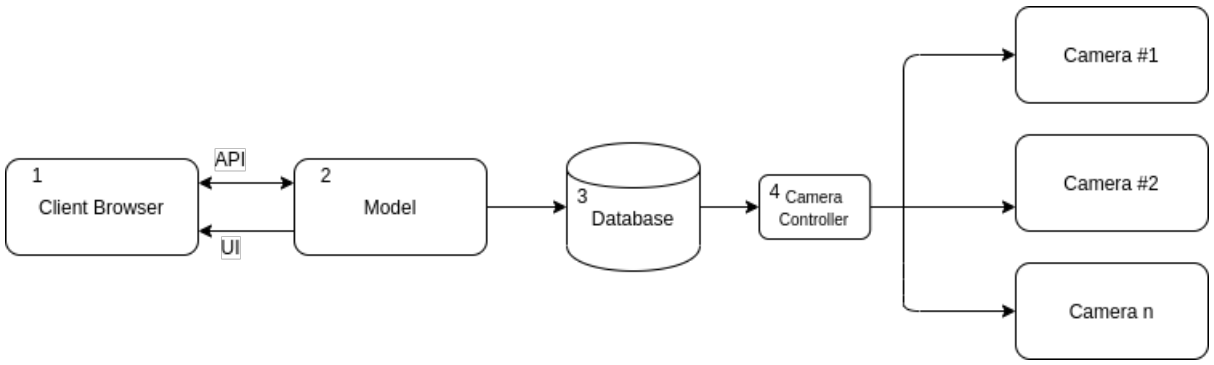

Figure 3.16: Architecture approach of framework

The first module is block 1 of Figure 3.16 is the client, which is responsible for permitting all of the interactions with the user and allow to see the cameras making the inference and see the boundary boxes and the labels, and the distance estimation as well.

The module responsible for controlling the model is block 2 of Figure 3.16, and it will be expanded in Figure 3.17. The input data is obtained from the data provided by cameras, and the output will be saved in the database. These outputs have already been defined in Section 3.3.

Block 3 of Figure 3.16 controls the usage flow of this framework and provides an abstraction layer to the usage of the database.

In block 4 of Figure 3.16 is the part responsible for connecting the other cameras via WIFI protocol and allows the system to connect another camera. The total amount of the camera is based on the hardware available for the tests.

Figure 3.17 shows how the model performs the inference process along the detection time, where this Figure shows the flow of the framework in new images. For example, the camera starts to stream, and to apply the object detection on the refereed frame, the dimension estimation network is called, and in another direction, the segmentation network is activated. After this step, it is possible to use the vehicle segmentation to detect vehicles along this way.

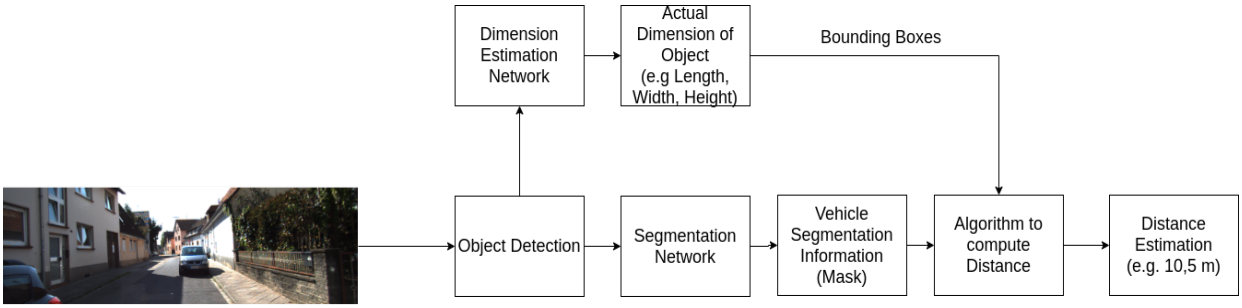

Figure 3.17: Architecture of framework based on multicameras perspective

Still using Figure 3.17, the next mutual box is the action to compute the algorithm to measure the distance of the object. In this box, the inputs are the relative position of each bounding box and the object's computed label.

This block diagram's last action is to show the user's predicted distance and save it into the relational database for future queries.

4 RESULTS

In this chapter, the results collected from the Chapter 4 will be discussed and analyzed. It is essential to freeze that only approach three from 3.3 was implemented into the framework because the goal was to work with multiple cameras and multi-view perspectives.

The algorithms and the simulations was performed in a computer with this follow configuration:

- Operational System Ubuntu 18.04
- CPU Intel core i7 7700HQ 2.80 GHz
- 32 GB memory RAM
- GPU Nvidia Geforce GTX 1050 Ti - 4 GB

The framework can perform the tests on the Audi test track, as shown in Figure 4.1, but in this work, only the proof of concept of the algorithms was performed.

Figure 4.1: Audi test track in eagle's view

The algorithm for object detection was performed over the parking lot of the company EFS GmbH as shown in Figure 4.2.

Figure 4.2: Position of the cars on the parking lot

4.1 DESCRIPTION OF THE TEST SCENARIO

The test scenario was built three times, using different perspectives. For the first test where it was necessary to use the camera, calibration perspective was made using only one camera, and the height of this camera does not matter to compute the distance, only a known distance, and dimensions of a known object to adjust this algorithm.

For the second scenario, some photos were taken and labeled with the known metrics to support the training step and return a useful result according to reality.

The last approach and used in this work to built the framework was used following some principles as the cameras should be mounted at the same level, the same horizontal position, the stream must be captured at the same time of the camera and sent to the same control center to process this data.

The used cameras were GoPro 5, which allow building a wireless network with many cameras.

4.2 RESULTS WITH CAMERA CALIBRATION

The results from approach one from 3.1 were implemented using Python 3.7 and Opencv3, and the position was computed based on camera calibration. Equation (4.1) is defined as the distance used in the camera calibration, using a measurer tape and a piece of paper (21.59 cm x 27.94 cm), and this was positioned 60.96 cm in front of the camera to take the photo.

$$F = \frac{P \cdot D}{W} \tag{4.1}$$

W is the piece of the paper's width, which is 27.94 cm, D is the distance from the piece of paper to the camera, and P is the paper's measure in pixels taken from the image. Applying the (4.1), the focal length (F) is 541.09 pixels.

The results achieved with this technique is shown in Table 4.1.

Table 4.1: Measurements achieved with camera calibration algorithm

Car	Measurements
#1	4.05
#2	10.67
#3	15.52
#4	19.55
#5	20.08

4.3 RESULTS WITH KNOWN MAP

In this section is detailed the results of the proposal with a known map. This approach was performed using the neural network from 3.2 and the data provided by KITTI Dataset [38]. It is essential to freeze. This approach was used only for the comparison method and not to be used on the final framework.

The results collected in this section are beneficial, as several companies and other universities are releasing the dataset as open source. There is a need to understand image manipulation, and in some cases, data from LIDAR and Radar.

This test's main idea was to get the information from the object detection performed with the algorithm using the boundary boxes approach, collect this known information as input in the neural network, and predict a distance from the camera until the object. Where in Figure 4.3 is shown an example of the known KITTI dataset, it serves just as motivation for this work. The final results were not performed over this dataset.

Figure 4.3: Output results from framework using single stereo camera and known map

For this approach, the output achieved from Table 3.2 was used for the first interaction and estimate the distance, as shown in Table 4.2. These results are similar to the reality because it uses a known map, and the measurement step of these structures was performed and the labeling part.

Table 4.2: Measurements achieved with camera and known map

Car	Measurements
#1	4.43
#2	10.98
#3	16.01
#4	18.99
#5	22.18

4.4 RESULTS WITH MULTI-CAMERAS AND PROPOSED FRAMEWORK

The proposed framework's algorithm predicts the objects of the whole scenario in 28 ms and has identified nine cars on the image, as shown in Figure 4.4 and in Table 4.4 is demonstrated the accuracy of each prediction for each vehicle. The output accuracies from the algorithm are shown in Table 4.5. The low accuracies are related to the partially occluded objects, such as car two and car 6.

The perspective of the distance this algorithm computed this using the Inverse Perspective Mapping (IPM) combined with the Yolo Algorithm, where the view of distance was fused on the last fully connected layer. In Table 4.3 is shown the results achieved from the perspective using

two cameras.

Table 4.3: Measurements achieved with multicameras

Car	Measurements
#1	4.40
#2	11.05
#3	16.01
#4	19.92
#5	24.08

In Figure 4.4 was used just the model that contains the object recognition and detection using a single image and not as a stream. This test was performed to see the algorithm's accuracy for this purpose and identify how far it can detect the objects in this scenario. It is necessary to note that this test previewed only five cars to detect, and the algorithm recognizes the whole vehicles in the scenario, each accuracy for each car of the test is available in Table 4.4.

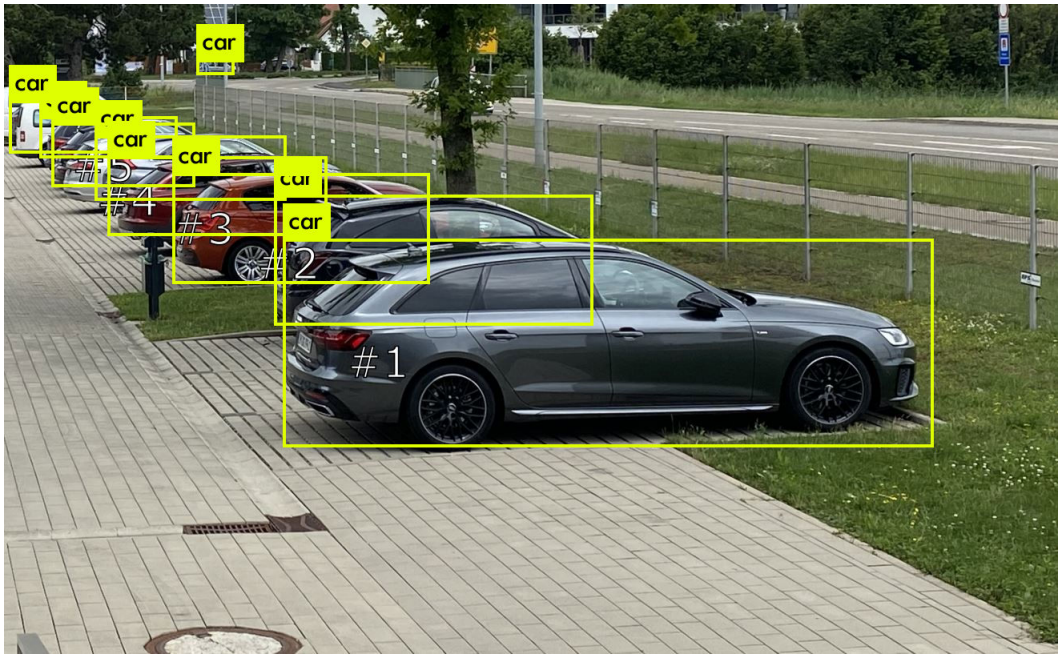

Figure 4.4: Output image with boundary boxes predictions

Table 4.4: Accuracy of the proposed framework in object detector and classification

Predicted Label	Accuracy
Car	91%
Car	31%
Car	96%
Car	94%
Car	95%
Car	98%
Car	47%
Car	97%
Car	98%

Figure 4.5 is shown the frontend of the application of this work. It was developed using Python and Javascript to allow the browser to communicate with the model. The Python module is composed of the libraries called sockets, and it permits the communication over many cameras and shares the stream information between the cameras and the command center. It was possible because the cameras used in the tests have an internal wireless network. Based on it, a script with a proxy function was developed to take care of this behavior.

The backend side of this project is described in Appendix I.1, where there is information about the pre-processing step, training step, object detection, and the frontend scenario as well. The network was built using the framework Pytorch [81]. This script permits the Darknet abstraction to perform object detection and object recognition, and with all of this information, the step to predict the distance was combined to show the output.

The camera 1 of the Figure 4.5 is used only to perform the boundary boxes detection, and camera two is used to perform the distance prediction. This solution was embedded in a module to work as a controller of this framework.

The experiments also led us to measure the multi-cameras method's rapidity by computing the number of frames treated per second. The average of frames per second through all the experiments is 45.57 frames per second, enough for real-time treatments.

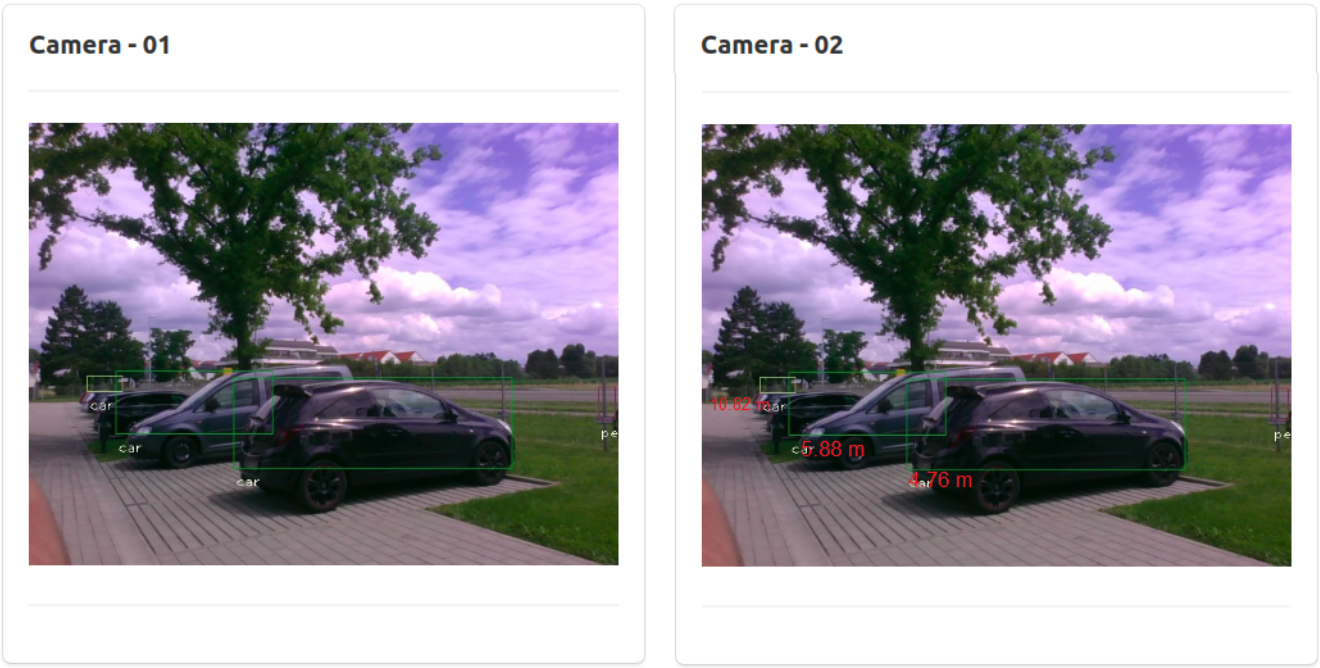

Figure 4.5: Output image with predictions

4.5 VALIDATION AND COMPARISON BETWEEN THE APPROACHES

For validation purpose, it was used a commercial laser measurement as shown in Figure 4.6, this model is known as Bosch DLE 40 Professional ®.

Figure 4.6: Commercial laser measurer

The current error rate is $\pm 1.5mm$, we repeated the measure three times and computed the mean and standard deviation. In Table 4.5 is shown the measurements with the camera positioned at 2.01 m from the ground. Figure 4.2 is established the position of the cars along the parking lot. The reason to measure three times is that the manual is written on sunny days and can bias the measurement.

Table 4.5: Measurements collected with a commercial measurer

Car	First measure (m)	Second measure (m)	Third measure (m)	Mean (m)
#1	4.25	4.47	4.51	4.41
#2	11.01	11.21	11.11	11.11
#3	16.12	16.35	16.26	16.24
#4	19.63	19.69	19.66	19.66
#5	23.08	23.18	23.01	23.09

To compare the differences between the approaches and the real distance computed with the tool from Figure 4.6. Table 4.6 was built to facilitate the visualization of these outputs.

Table 4.6: Comparison between the algorithms and real data

Car	Camera Calibration (m)	Known Map (m)	Multicameras (m)	Real distance (m)
#1	4.05	4.43	**4.40**	4.41
#2	10.67	10.98	**11.05**	11.11
#3	15.52	**16.01**	**16.01**	16.24
#4	**19.55**	18.99	19.92	19.66
#5	20.08	22.18	**24.08**	23.09

It is necessary to examine the error between the real value. For this, a Python script was implement using Pandas Library [82] to take care of the collected data. The error analysis was implemented to make it easier for the visualization of the best-applied technique.

The pre-processed samples are used as input for the algorithms under text, resulting in estimations of the true position value. Results are expressed in terms of the RMSE, given by (4.2):

$$\text{RMSE}(f, \hat{f}) = \sqrt{\frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (f_i - \hat{f}_i)^2}, \tag{4.2}$$

calculated for each estimator $\hat{f}_i$, referenced either from the measurement tool of Figure 4.6 true value f_i as read at the end of each measurement.

Figure 4.7 is shown a chart with the three proposed techniques using (4.2), where it is possible to see in specific points the multi-cameras approach was better than camera calibration.

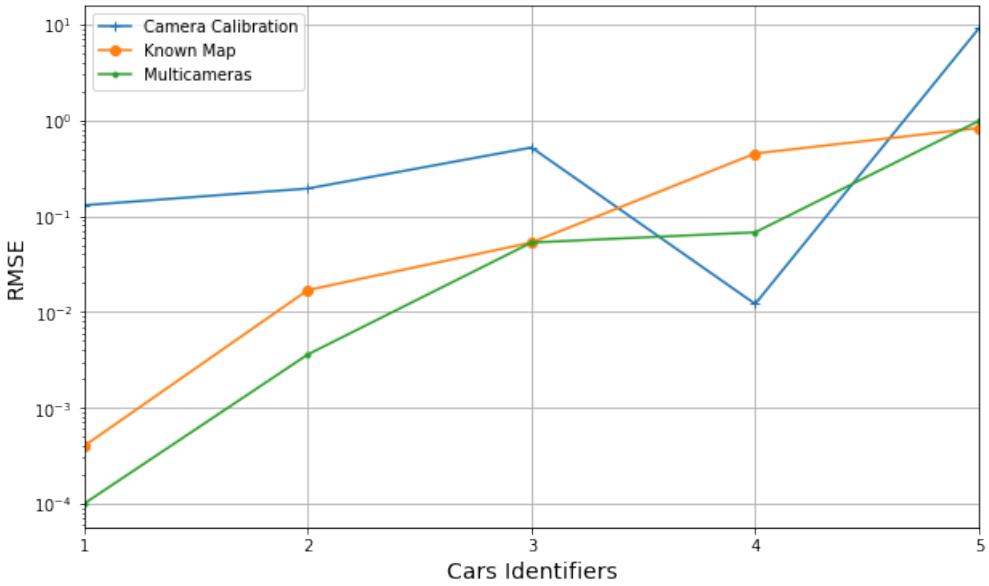

Figure 4.7: RMSE of estimated estimate position for each algorithm for different detected car, referenced to values measured by the commercial laser measurer

5 CONCLUSION

In the next years, autonomous vehicles will be a new reality, and at this moment, the topics regarding object detection and machine learning are the hot trends in the computer science domain. Furthermore, with this, it is necessary to improve computer vision methods to strengthen this related technology. Applying this algorithm to detect other classes of objects and perform distance measurement has improved accuracy.

A Real-time distance measurement method with multi-cameras for object detection on the roads is introduced in this work. The utilized method is based on using multi-cameras, two cameras mounted in the same horizontal position and displaced vertically by a predefined distance (the base). A vehicle detection method is performed first following two steps: hypothesis generation and hypothesis verification.

This work also compares several state-of-the-art techniques algorithms to perform distance measurement to choose the best and faster technique.

One of the big problems on camera arrays is putting in the correct angle and high level. A small filter to clear this threshold is necessary to be implemented and remove these noises. The part of calibration in the proposed technique 1 is a little bit difficult because it is needed to remember the tool error and the measurement error, and with this is possible to calibrate in the wrong way.

It was also proved that the multi-cameras perspective combined with a high order mathematical technique is better for this work, where it is possible to divide the work of the detection into two different or more cameras. However, for our motivation and based on the current literature, the camera angle is +30 degrees, which will allow us to cover a big part of the scenario.

5.1 FUTURE WORKS

According to all results collected in this work, it is possible to suggest the following approaches:

- Collect data and label data to perform predictions for a specific scenario.
- Combine data from multiple sensors, such as LIDAR or RADAR, using data fusion approaches to increase the measurement's accuracy.
- Apply other algorithms, like EfficienceDet or other novel algorithms for object detection and distance measurement.
- Use the proposed test in the real scenario because the algorithm was just performed in controlled sites.

- Perform other mathematical approaches to reduce the dimensionality of the data.

BIBLIOGRAPHY

1 BANSAL, P.; KOCKELMAN, K. M. Forecasting americans' long-term adoption of connected and autonomous vehicle technologies. *Transportation Research Part A: Policy and Practice*, Elsevier, v. 95, p. 49–63, 2017.

2 DEPARTMENT, G. S. Long term series: Road traffic accidents. *https://www.destatis.de/EN/Themes/Society-Environment/Traffic-Accidents/Tables/liste-traffic-accidents.html*, v. 6.

3 ADMINISTRATION, N. H. T. S. Summary of motor vehicle crashes. *https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812376*, v. 6.

4 BAYAT, B.; CRASTA, N.; CRESPI, A.; PASCOAL, A. M.; IJSPEERT, A. Environmental monitoring using autonomous vehicles: a survey of recent searching techniques. *Current opinion in biotechnology*, Elsevier, v. 45, p. 76–84, 2017.

5 RASOULI, A.; TSOTSOS, J. K. Autonomous vehicles that interact with pedestrians: A survey of theory and practice. *IEEE Transactions on Intelligent Transportation Systems*, IEEE, v. 21, n. 3, p. 900–918, 2019.

6 HOBERT, L.; FESTAG, A.; LLATSER, I.; ALTOMARE, L.; VISINTAINER, F.; KOVACS, A. Enhancements of v2x communication in support of cooperative autonomous driving. *IEEE communications magazine*, IEEE, v. 53, n. 12, p. 64–70, 2015.

7 XU, X.; FAN, C.-K. Autonomous vehicles, risk perceptions and insurance demand: An individual survey in china. *Transportation research part A: policy and practice*, Elsevier, v. 124, p. 549–556, 2019.

8 BONNEFON, J.-F.; SHARIFF, A.; RAHWAN, I. The social dilemma of autonomous vehicles. *Science*, American Association for the Advancement of Science, v. 352, n. 6293, p. 1573–1576, 2016.

9 Addabbo, T.; De Muro, S.; Falaschi, G.; Fort, A.; Landi, E.; Moretti, R.; Mugnaini, M.; Nicolelli, F.; Parri, L.; Tani, M.; Tesei, M.; Vignoli, V. An automatic battery recharge and condition monitoring system for autonomous drones. In: *2020 IEEE International Workshop on Metrology for Industry 4.0 IoT*. [S.l.: s.n.], 2020. p. 1–5.

10 LATEGAHN, H.; SCHREIBER, M.; ZIEGLER, J.; STILLER, C. Urban localization with camera and inertial measurement unit. *IEEE Intelligent Vehicles Symposium, Proceedings*, n. June, p. 719–724, 2013.

11 SANKARANARAYANAN, A. C.; VEERARAGHAVAN, A.; CHELLAPPA, R. Object detection, tracking and recognition for multiple smart cameras. *Proceedings of the IEEE*, v. 96, n. 10, p. 1606–1624, 2008. ISSN 00189219.

12 HARTLEY, R. I.; STURM, P. Triangulation. *Computer vision and image understanding*, Elsevier, v. 68, n. 2, p. 146–157, 1997.

13 UNLU, E.; ZENOU, E.; RIVIERE, N.; DUPOUY, P. E. Deep learning-based strategies for the detection and tracking of drones using several cameras. *IPSJ Transactions on Computer Vision and Applications*, IPSJ Transactions on Computer Vision and Applications, v. 11, n. 1, 2019. ISSN 18826695.

14 PEI, Z.; LI, Y.; MA, M.; LI, J.; LENG, C.; ZHANG, X.; ZHANG, Y. Occluded-object 3D reconstruction using camera array synthetic aperture imaging. *Sensors (Switzerland)*, v. 19, n. 3, p. 1–22, 2019. ISSN 14248220.

15 ZAARANE, A.; SLIMANI, I.; Al Okaishi, W.; ATOUF, I.; HAMDOUN, A. Distance measurement system for autonomous vehicles using stereo camera. *Array*, Elsevier Ltd, v. 5, n. May 2019, p. 100016, 2020. ISSN 25900056. Disponível em: <https://doi.org/10.1016/j.array.2020.100016>.

16 WU, H.; ZHANG, X.; STORY, B.; RAJAN, D. Accurate Vehicle Detection Using Multi-camera Data Fusion and Machine Learning. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, v. 2019-May, p. 3767–3771, 2019. ISSN 15206149.

17 ALI, A. A.; HUSSEIN, H. A. Distance estimation and vehicle position detection based on monocular camera. *Al-Sadiq International Conference on Multidisciplinary in IT and Communication Techniques Science and Applications, AIC-MITCSA 2016*, IEEE, n. 1, p. 20–23, 2016.

18 HÄNE, C.; HENG, L.; LEE, G. H.; FRAUNDORFER, F.; FURGALE, P.; SATTLER, T.; POLLEFEYS, M. 3D visual perception for self-driving cars using a multi-camera system: Calibration, mapping, localization, and obstacle detection. *Image and Vision Computing*, v. 68, n. August, p. 14–27, 2017. ISSN 02628856.

19 CUI, Z.; HENG, L.; YEO, Y. C.; GEIGER, A.; POLLEFEYS, M.; SATTLER, T. Real-time dense mapping for self-driving vehicles using fisheye cameras. *Proceedings - IEEE International Conference on Robotics and Automation*, v. 2019-May, p. 6087–6093, 2019. ISSN 10504729.

20 HUANG, L.; ZHE, T.; WU, J.; WU, Q.; PEI, C.; CHEN, D. Robust Inter-Vehicle Distance Estimation Method Based on Monocular Vision. *IEEE Access*, IEEE, v. 7, p. 46059–46070, 2019. ISSN 21693536.

21 BAO, J.; GU, Y.; HSU, L. T.; KAMIJO, S. Vehicle self-localization using 3D building map and stereo camera. *IEEE Intelligent Vehicles Symposium, Proceedings*, v. 2016-Augus, n. June, p. 927–932, 2016.

22 TANG, Z.; NAPHADE, M.; LIU, M. Y.; YANG, X.; BIRCHFIELD, S.; WANG, S.; KUMAR, R.; ANASTASIU, D.; HWANG, J. N. Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, v. 2019-June, p. 8789–8798, 2019. ISSN 10636919.

23 QI, S. H.; LI, J.; SUN, Z. P.; ZHANG, J. T.; SUN, Y. Distance Estimation of Monocular Based on Vehicle Pose Information. *Journal of Physics: Conference Series*, v. 1168, n. 3, 2019. ISSN 17426596.

24 WONGSAREE, P.; SINCHAI, S.; WARDKEIN, P.; KOSEEYAPORN, J. Distance Detection Technique Using Enhancing Inverse Perspective Mapping. *2018 3rd International Conference on Computer and Communication Systems, ICCCS 2018*, IEEE, p. 323–326, 2018.

25 PAN, X.; LIU, Z.; ZHANG, G. High-Accuracy Calibration of On-Site Multi-Vision Sensors Based on Flexible and Optimal 3D Field. *IEEE Access*, IEEE, v. 7, p. 159495–159506, 2019. ISSN 21693536.

26 LIN, C.; SU, F.; WANG, H.; GAO, J. A camera calibration method for obstacle distance measurement based on monocular vision. *Proceedings - 2014 4th International Conference on Communication Systems and Network Technologies, CSNT 2014*, IEEE, p. 1148–1151, 2014.

27 SIMON, M.; AMENDE, K.; KRAUS, A.; HONER, J.; SÄMANN, T.; KAULBERSCH, H.; MILZ, S.; GROSS, H. M. Complexer-YOLO: Real-Time 3D Object Detection and Tracking on Semantic Point Clouds. 2019. Disponível em: <http://arxiv.org/abs/1904.07537>.

28 KIM, J.; KIM, J.; CHO, J. An advanced object classification strategy using YOLO through camera and LiDAR sensor fusion. *2019, 13th International Conference on Signal Processing and Communication Systems, ICSPCS 2019 - Proceedings*, IEEE, p. 1–5, 2019.

29 SIMON, M.; MILZ, S.; AMENDE, K.; GROSS, H. M. Complex-YOLO: An euler-region-proposal for real-time 3D object detection on point clouds. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 11129 LNCS, p. 197–209, 2019. ISSN 16113349.

30 OLIVEIRA, M.; SANTOS, V.; SAPPA, A. D. Multimodal inverse perspective mapping. *Information Fusion*, Elsevier B.V., v. 24, p. 108–121, 2015. ISSN 15662535. Disponível em: <http://dx.doi.org/10.1016/j.inffus.2014.09.003>.

31 SALMAN, Y. D.; KU-MAHAMUD, K. R.; KAMIOKA, E. Distance Measurement for Self-Driving Cars Using Stereo Camera. *Proceeding of the 6Th International Conference of Computing & Informations*, n. 105, p. 235–242, 2017.

32 RANGESH, A.; TRIVEDI, M. M. No Blind Spots: Full-Surround Multi-Object Tracking for Autonomous Vehicles Using Cameras and LiDARs. *IEEE Transactions on Intelligent Vehicles*, v. 4, n. 4, p. 588–599, 2019. ISSN 2379-8858.

33 TRAM, V. T. B.; YOO, M. Vehicle-To-Vehicle Distance Estimation Using a Low-Resolution Camera Based on Visible Light Communications. *IEEE Access*, IEEE, v. 6, p. 4521–4527, 2018. ISSN 21693536.

34 HOFMANN, C.; PARTICKE, F.; HILLER, M.; THIELECKE, J. Object detection, classification and localization by infrastructural stereo cameras. *VISIGRAPP 2019 - Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, v. 5, p. 808–815, 2019.

35 ADMINISTRATION, N. H. T. S. et al. Preliminary statement of policy concerning automated vehicles. *Washington, DC*, p. 1–14, 2013.

36 KRASNER, G.; KATZ, E. Automatic parking identification and vehicle guidance with road awareness. In: IEEE. *2016 IEEE International Conference on the Science of Electrical Engineering (ICSEE)*. [S.l.], 2016. p. 1–5.

37 SCHÖNING, V.; KATZWINKEL, R.; WUTTKE, U.; SCHWITTERS, F.; ROHLFS, M.; SCHULER, T. Der parklenkassistent" park assist" von volkswagen/the volkswagen" park assist". *VDI-Berichte*, n. 1960, 2006.

38 GEIGER, A.; LENZ, P.; STILLER, C.; URTASUN, R. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, Sage Publications Sage UK: London, England, v. 32, n. 11, p. 1231–1237, 2013.

39 CAESAR, H.; BANKITI, V.; LANG, A. H.; VORA, S.; LIONG, V. E.; XU, Q.; KRISHNAN, A.; PAN, Y.; BALDAN, G.; BEIJBOM, O. nuscenes: A multimodal dataset for autonomous driving. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2020. p. 11621–11631.

40 CUTLER, K.-M. How many american cities are preparing for the arrival of self-driving cars? not many. *Techcrunch. com, http://bit. ly/CitiesUnprepared*, v. 18, 2015.

41 ORGANIZATION, W. H. et al. World report on road traffic injury prevention. In: *World report on road traffic injury prevention*. [S.l.: s.n.], 2015. p. 217–217.

42 ROSS, P. E. The audi a8: the world's first production car to achieve level 3 autonomy. *IEEE Spectrum*, v. 1, 2017.

43 KOCIĆ, J.; JOVIČIĆ, N.; DRNDAREVIĆ, V. Sensors and sensor fusion in autonomous vehicles. In: IEEE. *2018 26th Telecommunications Forum (TELFOR)*. [S.l.], 2018. p. 420–425.

44 KATO, S.; TAKEUCHI, E.; ISHIGURO, Y.; NINOMIYA, Y.; TAKEDA, K.; HAMADA, T. An open approach to autonomous vehicles. *IEEE Micro*, IEEE, v. 35, n. 6, p. 60–68, 2015.

45 VARGHESE, J. Z.; BOONE, R. G. et al. Overview of autonomous vehicle sensors and systems. In: *International Conference on Operations Excellence and Service Engineering*. [S.l.: s.n.], 2015. p. 178–191.

46 KRASNIQI, X.; HAJRIZI, E. Use of iot technology to drive the automotive industry from connected to full autonomous vehicles. *IFAC-PapersOnLine*, Elsevier, v. 49, n. 29, p. 269–274, 2016.

47 Zhang, Z. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 22, n. 11, p. 1330–1334, 2000.

48 FLIR. *FLIR Showcases Next Generation, All-Weather Thermal Camera on Self Driving Cars*. jan. 2019. <https://www.flir.com/news-center/camera-cores--components/all-weather-thermal-cameras-coming-to-a-self-driving-car-near-you/>.

49 PARO, J. A.; NAZARELI, R.; GURJALA, A.; BERGER, A.; LEE, G. K. Video-based self-review: comparing google glass and gopro technologies. *Annals of plastic surgery*, LWW, v. 74, p. S71–S74, 2015.

50 ARIYUR, K.; ENNS, D.; LOMMEL, P. *Collision avoidance involving radar feedback*. [S.l.]: Google Patents, mar. 16 2006. US Patent App. 10/941,535.

51 STEVENSON, R. Long-distance car radar. *IEEE Spectrum*, 2011. Available at <https://spectrum.ieee.org/transportation/advanced-cars/longditance-car-radar>, accessed at 17 June 2020.

52 INSTITUTE, F. Radar sensor module to bring added safety to autonomous driving. *Fraunhofer Institute*, 2019. Available at <https://www.fraunhofer.de/en/press/research-news/2019/june/radar-sensor-module-to-bring-added-safety-to-autonomous-driving.html>, accessed at 17 June 2020.

53 GAO, H.; CHENG, B.; WANG, J.; LI, K.; ZHAO, J.; LI, D. Object classification using cnn-based fusion of vision and lidar in autonomous vehicle environment. *IEEE Transactions on Industrial Informatics*, IEEE, v. 14, n. 9, p. 4224–4231, 2018.

54 LIM, H. S. M.; TAEIHAGH, A. Algorithmic decision-making in avs: Understanding ethical and technical concerns for smart cities. *Sustainability*, Multidisciplinary Digital Publishing Institute, v. 11, p. 5791, 10 2019. ISSN 2071-1050. Disponível em: <https://www.mdpi.com/2071-1050/11/20/5791>.

55 DOSOVITSKIY, A.; ROS, G.; CODEVILLA, F.; LOPEZ, A.; KOLTUN, V. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

56 DWORAK, D.; CIEPIELA, F.; DERBISZ, J.; IZZAT, I.; KOMORKIEWICZ, M.; WÓJCIK, M. Performance of lidar object detection deep learning architectures based on artificially generated point cloud data from carla simulator. In: IEEE. *2019 24th International Conference on Methods and Models in Automation and Robotics (MMAR)*. [S.l.], 2019. p. 600–605.

57 GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. [S.l.]: MIT press, 2016.

58 COSTA, J. P. C. L da. Autonomous vehicles by machine learning. 2019.

59 LECUN, Y.; BENGIO, Y. et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, v. 3361, n. 10, p. 1995, 1995.

60 YANG, Y.; SAUTIÈRE, G.; RYU, J. J.; COHEN, T. S. Feedback recurrent autoencoder. In: IEEE. *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.], 2020. p. 3347–3351.

61 PASCANU, R.; MIKOLOV, T.; BENGIO, Y. On the difficulty of training recurrent neural networks. In: *International conference on machine learning*. [S.l.: s.n.], 2013. p. 1310–1318.

62 Huang, L.; Zhe, T.; Wu, J.; Wu, Q.; Pei, C.; Chen, D. Robust inter-vehicle distance estimation method based on monocular vision. *IEEE Access*, v. 7, p. 46059–46070, 2019.

63 ZHU, H.; LI, Y.; LIU, X.; YIN, X.; SHAO, Y.; QIAN, Y.; TAN, J. Camera calibration from very few images based on soft constraint optimization. *Journal of the Franklin Institute*, Elsevier, v. 357, n. 4, p. 2561–2584, 2020.

64 KAEHLER, A.; BRADSKI, G. *Learning OpenCV 3: computer vision in C++ with the OpenCV library*. [S.l.]: " O'Reilly Media, Inc.", 2016.

65 FORSYTH, D. A.; PONCE, J. *Computer vision: a modern approach*. [S.l.]: Prentice Hall Professional Technical Reference, 2002.

66 WANG, P.; SHI, T.; ZOU, C.; XIN, L.; CHAN, C.-Y. A data driven method of feedforward compensator optimization for autonomous vehicle control. In: IEEE. *2019 IEEE Intelligent Vehicles Symposium (IV)*. [S.l.], 2019. p. 2012–2017.

67 PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V. et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, JMLR. org, v. 12, p. 2825–2830, 2011.

68 LIU, Q.; CHENG, J.; LU, H.; MA, S. Distance based kernel pca image reconstruction. In: IEEE. *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* [S.l.], 2004. v. 3, p. 670–673.

69 HALKO, N.; MARTINSSON, P. G.; TROPP, J. A. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, Society for Industrial Applied Mathematics (SIAM), v. 53, n. 2, p. 217–288, Jan 2011. ISSN 1095-7200. Disponível em: <http://dx.doi.org/10.1137/090771806>.

70 CAMBRIDGE, U. Introduction to information retrieval. 2009.

71 REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 779–788.

72 REDMON, J.; FARHADI, A. Yolo9000: better, faster, stronger. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 7263–7271.

73 ESS, A.; SCHINDLER, K.; LEIBE, B.; GOOL, L. V. Object detection and tracking for autonomous navigation in dynamic environments. *The International Journal of Robotics Research*, SAGE Publications Sage UK: London, England, v. 29, n. 14, p. 1707–1725, 2010.

74 KRASIN, I.; DUERIG, T.; ALLDRIN, N.; FERRARI, V.; ABU-EL-HAIJA, S.; KUZNETSOVA, A.; ROM, H.; UIJLINGS, J.; POPOV, S.; VEIT, A. et al. Openimages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://github. com/openimages*, v. 2, n. 3, p. 2–3, 2017.

75 REDMON, J. *Darknet: Open source neural networks in c*. 2013.

76 CAO, Y.-T.; WANG, J.-M.; SUN, Y.-K.; DUAN, X.-J. Circle marker based distance measurement using a single camera. *Lecture Notes on Software Engineering*, IACSIT Press, v. 1, n. 4, p. 376, 2013.

77 MAYER, N.; ILG, E.; HAUSSER, P.; FISCHER, P.; CREMERS, D.; DOSOVITSKIY, A.; BROX, T. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 4040–4048.

78 TZUTALIN, D. Labelimg (2015). *GitHub repository https://github. com/tzutalin/labelImg*, v. 6.

79 TUOHY, S.; O'CUALAIN, D.; JONES, E.; GLAVIN, M. Distance determination for an automobile environment using inverse perspective mapping in OpenCV. *IET Conference Publications*, v. 2010, n. 566 CP, p. 100–105, 2010.

80 Ali, A. A.; Hussein, H. A. Distance estimation and vehicle position detection based on monocular camera. In: *2016 Al-Sadeq International Conference on Multidisciplinary in IT and Communication Science and Applications (AIC-MITCSA)*. [S.l.: s.n.], 2016. p. 1–4.

81 PASZKE, A.; GROSS, S.; MASSA, F.; LERER, A.; BRADBURY, J.; CHANAN, G.; KILLEEN, T.; LIN, Z.; GIMELSHEIN, N.; ANTIGA, L. et al. Pytorch: An imperative style, high-performance deep learning library. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2019. p. 8026–8037.

82 MCKINNEY, W. et al. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, Seattle, v. 14, n. 9, 2011.

APPENDICES

I.1 CODE TO CONTROL THE APP

]

```
# Flask utils
from flask import Flask, redirect, url_for, request,
    render_template, Response
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer
from camera import ObjectDetection

app = Flask(__name__)
@app.route("/")
def main():
    return render_template("index.html")

def gen(camera):
    while True:
        frame = camera.main()
        if frame != "":
            yield (b --frame\r\n
                b Content-Type: image/jpeg\r\n\r\n  + frame +
                   b \r\n\r\n )

@app.route( /video_feed )
def video_feed():
    id = 0
    return Response(gen(ObjectDetection(id)), mimetype=
       multipart/x-mixed-replace; boundary=frame )

def simulate(camera):
    while True:
        frame = camera.main()
        if frame != "":
            yield (b --frame\r\n
                b Content-Type: image/jpeg\r\n\r\n  + frame +
                   b \r\n\r\n )

@app.route( /video_simulate )
def video_simulate():
```

```
33      id = 1
34      return Response(gen(ObjectDetection(id)), mimetype=
            multipart/x-mixed-replace; boundary=frame )
35
36
37  if __name__ ==  __main__ :
38      # Serve the app with gevent
39      app.run(host= 0.0.0.0 , threaded=True, debug = True)
```

I.2 CODE TO DETECT THE BOUNDING BOXES

```
from __future__ import division

import torch
import random

import numpy as np
import cv2

def confidence_filter(result, confidence):
    conf_mask = (result[:,:,4] > confidence).float().unsqueeze
       (2)
    result = result*conf_mask

    return result

def confidence_filter_cls(result, confidence):
    max_scores = torch.max(result[:,:,5:25], 2)[0]
    res = torch.cat((result, max_scores),2)
    print(res.shape)


    cond_1 = (res[:,:,4] > confidence).float()
    cond_2 = (res[:,:,25] > 0.995).float()

    conf = cond_1 + cond_2
    conf = torch.clamp(conf, 0.0, 1.0)
    conf = conf.unsqueeze(2)
    result = result*conf
    return result



def get_abs_coord(box):
    box[2], box[3] = abs(box[2]), abs(box[3])
    x1 = (box[0] - box[2]/2) - 1
    y1 = (box[1] - box[3]/2) - 1
    x2 = (box[0] + box[2]/2) - 1
    y2 = (box[1] + box[3]/2) - 1
```

```
38      return x1, y1, x2, y2
39
40
41
42  def sanity_fix(box):
43      if (box[0] > box[2]):
44          box[0], box[2] = box[2], box[0]
45
46      if (box[1] >  box[3]):
47          box[1], box[3] = box[3], box[1]
48
49      return box
50
51  def bbox_iou(box1, box2):
52
53
54
55      b1_x1, b1_y1, b1_x2, b1_y2 = box1[:,0], box1[:,1], box1
           [:,2], box1[:,3]
56      b2_x1, b2_y1, b2_x2, b2_y2 = box2[:,0], box2[:,1], box2
           [:,2], box2[:,3]
57
58
59      inter_rect_x1 =  torch.max(b1_x1, b2_x1)
60      inter_rect_y1 =  torch.max(b1_y1, b2_y1)
61      inter_rect_x2 =  torch.min(b1_x2, b2_x2)
62      inter_rect_y2 =  torch.min(b1_y2, b2_y2)
63
64
65      if torch.cuda.is_available():
66              inter_area = torch.max(inter_rect_x2 -
                   inter_rect_x1 + 1,torch.zeros(inter_rect_x2.
                   shape).cuda())*torch.max(inter_rect_y2 -
                   inter_rect_y1 + 1, torch.zeros(inter_rect_x2.
                   shape).cuda())
67      else:
68              inter_area = torch.max(inter_rect_x2 -
                   inter_rect_x1 + 1,torch.zeros(inter_rect_x2.
                   shape))*torch.max(inter_rect_y2 - inter_rect_y1
                    + 1, torch.zeros(inter_rect_x2.shape))
69
```

```
70
71      b1_area = (b1_x2 - b1_x1 + 1)*(b1_y2 - b1_y1 + 1)
72      b2_area = (b2_x2 - b2_x1 + 1)*(b2_y2 - b2_y1 + 1)
73
74      iou = inter_area / (b1_area + b2_area - inter_area)
75
76      return iou
77
78
79  def pred_corner_coord(prediction):
80
81      ind_nz = torch.nonzero(prediction[:,:,4]).transpose(0,1).
            contiguous()
82
83      box = prediction[ind_nz[0], ind_nz[1]]
84
85
86      box_a = box.new(box.shape)
87      box_a[:,0] = (box[:,0] - box[:,2]/2)
88      box_a[:,1] = (box[:,1] - box[:,3]/2)
89      box_a[:,2] = (box[:,0] + box[:,2]/2)
90      box_a[:,3] = (box[:,1] + box[:,3]/2)
91      box[:,:4] = box_a[:,:4]
92
93      prediction[ind_nz[0], ind_nz[1]] = box
94
95      return prediction
96
97
98
99
100 def write(x, batches, results, colors, classes):
101     c1 = tuple(x[1:3].int())
102     c2 = tuple(x[3:5].int())
103     img = results[int(x[0])]
104     cls = int(x[-1])
105     label = "{0}".format(classes[cls])
106     color = random.choice(colors)
107     cv2.rectangle(img, c1, c2,color, 1)
108     t_size = cv2.getTextSize(label, cv2.FONT_HERSHEY_PLAIN, 1
            , 1)[0]
```

```
109     c2 = c1[0] + t_size[0] + 3, c1[1] + t_size[1] + 4
110     cv2.rectangle(img, c1, c2,color, -1)
111     cv2.putText(img, label, (c1[0], c1[1] + t_size[1] + 4),
           cv2.FONT_HERSHEY_PLAIN, 1, [225,255,255], 1);
112     return img
```

I.3 CODE TO CONTROL THE CAMERA

```
1
2  from preprocess import letterbox_image
3
4  from darknet import Darknet
5
6  from imutils.video import WebcamVideoStream,FPS
7
8  import numpy as np
9  import torch.nn as nn
10 from torch.autograd import Variable
11 import torch,cv2,random,os,time
12 import pickle as pkl
13 import argparse
14 import threading, queue
15 from torch.multiprocessing import Pool, Process,
      set_start_method
16 from util import write_results, load_classes
17 torch.multiprocessing.set_start_method( spawn )
18
19 def image_preparation(img, inp_dim):
20
21
22
23     original_im = img
24     dim = orig_im.shape[1], orig_im.shape[0]
25     img = (letter_image(orig_im, (input_dim, input_dim)))
26     img_ = img[:, :, ::-1].transpose((2, 0, 1)).copy()
27     img_ = torch.from_numpy(original_im).float().div(255.0).
          unsqueeze(0)
28     return img_, orig_im, dim
```

```
29
30 labels = {}
31 b_boxe = {}
32 def write(bboxes, img, classes, colors):
33
34     x = b_boxes
35     b_boxes = b_boxes[1:5]
36     b_boxes = b_boxes.cpu().data.numpy()
37     b_boxes = b_boxes.astype(int)
38     b_boxes.update({"bbox":b_boxes.tolist()})
39
40     b_boxes = torch.from_numpy(b_boxes)
41     cls = int(x[-1])
42     label = "{0}".format(classes[cls])
43     labels.update({"Current Object":label})
44     color = random.choice(colors)
45     img = cv2.rectangle(img, (b_boxes[0],b_boxes[1]),(b_boxes
          [2],b_boxes[3]), color, 1)
46
47     img = cv2.putText(img, label, (b_boxes[0]+2,b_boxes[3]+20)
          , cv2.FONT_HERSHEY_PLAIN, 1, [225, 255, 255], 1)
48     return img
49
50 class ObjectDetection:
51     def __init__(self, id):
52
53         self.cap = cv2.VideoCapture(0)
54         self.cap = WebcamVideoStream(src = id).start()
55         self.cfgfile = "cfg/yolov3.cfg"
56
57         self.weightsfile = "yolov3.weights"
58
59         self.conf = float(0.5)
60         self.nms_trhesh = round(0.4)
61         self.amount_classes = 80
62         self.classes = load_classes( data/coco.names )
63         self.colors = pkl.load(open("pallete", "rb"))
64         self.model = Darknet(self.cfgfile)
65         self.CUDA = torch.cuda.is_available()
66         self.model.load_weights(self.weightsfile)
67         self.model.net_info["height"] = 160
```

```
68          self.inp_dim = int(self.model.net_info["height"])
69          self.width = 640
70          self.height = 480
71          print("Loading network.....")
72          if self.CUDA:
73              self.model.cuda()
74          print("Network successfully loaded")
75          assert self.inp_dim % 32 == 0
76          assert self.inp_dim > 32
77          self.model.eval()
78
79      def main(self):
80          q = queue.Queue()
81          def frame_render(queue_from_cam):
82              ret, frame = self.cap.read()
83              frame = cv2.resize(frame,(self.wi, self.hei))
84              queue_from_cam.put(frame)
85          cam = threading.Thread(target=frame_render, args=(q,))
86          cam.start()
87          cam.join()
88          frame = q.get()
89          q.task_done()
90          fps = FPS().start()
91
92          image, orig_image, dim = prep_image(frame, self.
                inp_dim)
93          im_dim = torch.FloatTensor(dim).repeat(1,2)
94          if self.CUDA:
95              im_dim = im_dim.cuda()
96              img = img.cuda()
97
98          out_image = self.model(Variable(img), self.CUDA)
99          out_image = write_results(output, self.confidence,
                self.num_classes, nms = True, nms_conf = self.
                nms_thesh)
100         output = output.type(torch.half)
101         if list(output.size()) == [1,86]:
102             #do nothing
103         else:
104             out_image[:,1:5] = torch.clamp(out_image[:,1:5],
                    0.0, float(self.inp_dim))/self.inp_dim
```

```
105
106
107                 out_image[:,[1,3]] *= frame.shape[1]
108                 out_image[:,[2,4]] *= frame.shape[0]
109                 list(map(lambda x: write(x, frame, self.classes,
                        self.colors),out_image))
110
111                 x,y,w,h = b_boxes["b_box"][0],b_boxes["bbox"][1],
                        b_boxes["bbox"][2], b_boxes["bbox"][3]
112                 dist = (2 * 3.14 * 180) / (w + h * 360)
113                 dist = round(distance * 2.54, 1)
114                 fb = ("{}".format(labels["Current Object"])+ " " +
                        "is"+" at {} ".format(round(dist))+"cm")
115
116                 print(feedback)
117
118
119
120
121
122             fps.update()
123             fps.stop()
124             print("[INFO] elasped time: {:.2f}".format(fps.elapsed
                    ()))
125             print("[INFO] approx. FPS: {:.1f}".format(fps.fps()))
126             ret, jpeg = cv2.imencode( .jpg , frame)
127             return jpeg.tostring()
```

I.4 ABSTRACTION OF DARKNET IN PYTORCH

```
1 from __future__ import division
2
3 import torch
4 import torch.nn as nn
5 import torch.nn.functional as F
6 from torch.autograd import Variable
7 import numpy as np
8 import cv2
```

```
import matplotlib.pyplot as plt
from util import count_parameters as count
from util import convert2cpu as cpu
from util import predict_transform

class test_net(nn.Module):
    def __init__(self, num_layers, input_size):
        super(test_net, self).__init__()
        self.num_layers= num_layers
        self.linear_1 = nn.Linear(input_size, 5)
        self.middle = nn.ModuleList([nn.Linear(5,5) for x in
           range(num_layers)])
        self.output = nn.Linear(5,2)

    def forward(self, x):
        x = x.view(-1)
        fwd = nn.Sequential(self.linear_1, *self.middle, self.
           output)
        return fwd(x)

def get_test_input():
    img = cv2.imread("dog-cycle-car.png")
    img = cv2.resize(img, (416,416))
    img_ =  img[:,:,::-1].transpose((2,0,1))
    img_ = img_[np.newaxis,:,:,:]/255.0
    img_ = torch.from_numpy(img_).float()
    img_ = Variable(img_)
    return img_


def parse_cfg(cfgfile):
    """
    Takes a configuration file

    Returns a list of blocks. Each blocks describes a block in
        the neural
    network to be built. Block is represented as a dictionary
       in the list

    """
    file = open(cfgfile,  r )
```

```
    lines = file.read().split( \n )
    lines = [x for x in lines if len(x) > 0]
    lines = [x for x in lines if x[0] !=  # ]
    lines = [x.rstrip().lstrip() for x in lines]


    block = {}
    blocks = []

    for line in lines:
        if line[0] == "[":
            if len(block) != 0:
                blocks.append(block)
                block = {}
            block["type"] = line[1:-1].rstrip()
        else:
            key,value = line.split("=")
            block[key.rstrip()] = value.lstrip()
    blocks.append(block)

    return blocks


import pickle as pkl

class MaxPoolStride1(nn.Module):
    def __init__(self, kernel_size):
        super(MaxPoolStride1, self).__init__()
        self.kernel_size = kernel_size
        self.pad = kernel_size - 1

    def forward(self, x):
        padded_x = F.pad(x, (0,self.pad,0,self.pad), mode="
           replicate")
        pooled_x = nn.MaxPool2d(self.kernel_size, self.pad)(
           padded_x)
        return pooled_x


class EmptyLayer(nn.Module):
    def __init__(self):
```

```
85          super(EmptyLayer, self).__init__()
86
87
88  class DetectionLayer(nn.Module):
89      def __init__(self, anchors):
90          super(DetectionLayer, self).__init__()
91          self.anchors = anchors
92
93      def forward(self, x, inp_dim, num_classes, confidence):
94          x = x.data
95          global CUDA
96          prediction = x
97          prediction = predict_transform(prediction, inp_dim,
                self.anchors, num_classes, confidence, CUDA)
98          return prediction
99
100
101
102
103
104 class Upsample(nn.Module):
105     def __init__(self, stride=2):
106         super(Upsample, self).__init__()
107         self.stride = stride
108
109     def forward(self, x):
110         stride = self.stride
111         assert(x.data.dim() == 4)
112         B = x.data.size(0)
113         C = x.data.size(1)
114         H = x.data.size(2)
115         W = x.data.size(3)
116         ws = stride
117         hs = stride
118         x = x.view(B, C, H, 1, W, 1).expand(B, C, H, stride, W
                , stride).contiguous().view(B, C, H*stride, W*
                stride)
119         return x
120 #
121
122 class ReOrgLayer(nn.Module):
```

```
123     def __init__(self, stride = 2):
124         super(ReOrgLayer, self).__init__()
125         self.stride= stride
126 
127     def forward(self,x):
128         assert(x.data.dim() == 4)
129         B,C,H,W = x.data.shape
130         hs = self.stride
131         ws = self.stride
132         assert(H % hs == 0),  "The stride " + str(self.stride)
                + " is not a proper divisor of height " + str(H)
133         assert(W % ws == 0),  "The stride " + str(self.stride)
                + " is not a proper divisor of height " + str(W)
134         x = x.view(B,C, H // hs, hs, W // ws, ws).transpose
               (-2,-3).contiguous()
135         x = x.view(B,C, H // hs * W // ws, hs, ws)
136         x = x.view(B,C, H // hs * W // ws, hs*ws).transpose
               (-1,-2).contiguous()
137         x = x.view(B, C, ws*hs, H // ws, W // ws).transpose
               (1,2).contiguous()
138         x = x.view(B, C*ws*hs, H // ws, W // ws)
139         return x
140 
141 
142 def create_modules(blocks):
143     net_info = blocks[0]
144 
145     module_list = nn.ModuleList()
146 
147     index = 0
148 
149 
150     prev_filters = 3
151 
152     output_filters = []
153 
154     for x in blocks:
155         module = nn.Sequential()
156 
157         if (x["type"] == "net"):
158             continue
```

```
159
160
161         if (x["type"] == "convolutional"):
162
163             activation = x["activation"]
164             try:
165                 batch_normalize = int(x["batch_normalize"])
166                 bias = False
167             except:
168                 batch_normalize = 0
169                 bias = True
170
171             filters= int(x["filters"])
172             padding = int(x["pad"])
173             kernel_size = int(x["size"])
174             stride = int(x["stride"])
175
176             if padding:
177                 pad = (kernel_size - 1) // 2
178             else:
179                 pad = 0
180
181
182             conv = nn.Conv2d(prev_filters, filters,
                   kernel_size, stride, pad, bias = bias)
183             module.add_module("conv_{0}".format(index), conv)
184
185
186             if batch_normalize:
187                 bn = nn.BatchNorm2d(filters)
188                 module.add_module("batch_norm_{0}".format(
                       index), bn)
189
190
191
192
193             if activation == "leaky":
194                 activn = nn.LeakyReLU(0.1, inplace = True)
195                 module.add_module("leaky_{0}".format(index),
                       activn)
196
```

```
197
198
199
200
201
202         elif (x["type"] == "upsample"):
203             stride = int(x["stride"])
204
205             upsample = nn.Upsample(scale_factor = 2, mode = "
                    nearest")
206             module.add_module("upsample_{}".format(index),
                    upsample)
207
208
209         elif (x["type"] == "route"):
210             x["layers"] = x["layers"].split( , )
211
212
213             start = int(x["layers"][0])
214
215
216             try:
217                 end = int(x["layers"][1])
218             except:
219                 end = 0
220
221
222
223
224             if start > 0:
225                 start = start - index
226
227             if end > 0:
228                 end = end - index
229
230
231             route = EmptyLayer()
232             module.add_module("route_{0}".format(index), route
                    )
233
234
```

```
235
236             if end < 0:
237                 filters = output_filters[index + start] +
                        output_filters[index + end]
238             else:
239                 filters= output_filters[index + start]
240
241
242
243
244         elif x["type"] == "shortcut":
245             from_ = int(x["from"])
246             shortcut = EmptyLayer()
247             module.add_module("shortcut_{}".format(index),
                    shortcut)
248
249
250         elif x["type"] == "maxpool":
251             stride = int(x["stride"])
252             size = int(x["size"])
253             if stride != 1:
254                 maxpool = nn.MaxPool2d(size, stride)
255             else:
256                 maxpool = MaxPoolStride1(size)
257
258             module.add_module("maxpool_{}".format(index),
                    maxpool)
259
260
261         elif x["type"] == "yolo":
262             mask = x["mask"].split(",")
263             mask = [int(x) for x in mask]
264
265
266             anchors = x["anchors"].split(",")
267             anchors = [int(a) for a in anchors]
268             anchors = [(anchors[i], anchors[i+1]) for i in
                    range(0, len(anchors),2)]
269             anchors = [anchors[i] for i in mask]
270
271             detection = DetectionLayer(anchors)
```

```
272                 module.add_module("Detection_{}".format(index),
                        detection)
273
274
275
276             else:
277                 print("Something I dunno")
278                 assert False
279
280
281             module_list.append(module)
282             prev_filters = filters
283             output_filters.append(filters)
284             index += 1
285
286
287         return (net_info, module_list)
288
289
290
291 class Darknet(nn.Module):
292         def __init__(self, cfgfile):
293             super(Darknet, self).__init__()
294             self.blocks = parse_cfg(cfgfile)
295             self.net_info, self.module_list = create_modules(self.
                    blocks)
296             self.header = torch.IntTensor([0,0,0,0])
297             self.seen = 0
298
299
300
301         def get_blocks(self):
302             return self.blocks
303
304         def get_module_list(self):
305             return self.module_list
306
307
308         def forward(self, x, CUDA):
309             detections = []
310             modules = self.blocks[1:]
```

```
311         outputs = {}
312
313
314         write = 0
315         for i in range(len(modules)):
316
317             module_type = (modules[i]["type"])
318             if module_type == "convolutional" or module_type
                   == "upsample" or module_type == "maxpool":
319
320                 x = self.module_list[i](x)
321                 outputs[i] = x
322
323
324             elif module_type == "route":
325                 layers = modules[i]["layers"]
326                 layers = [int(a) for a in layers]
327
328                 if (layers[0]) > 0:
329                     layers[0] = layers[0] - i
330
331                 if len(layers) == 1:
332                     x = outputs[i + (layers[0])]
333
334                 else:
335                     if (layers[1]) > 0:
336                         layers[1] = layers[1] - i
337
338                     map1 = outputs[i + layers[0]]
339                     map2 = outputs[i + layers[1]]
340
341
342                     x = torch.cat((map1, map2), 1)
343                 outputs[i] = x
344
345             elif  module_type == "shortcut":
346                 from_ = int(modules[i]["from"])
347                 x = outputs[i-1] + outputs[i+from_]
348                 outputs[i] = x
349
350
```

```
351
352             elif module_type ==  yolo :
353
354                 anchors = self.module_list[i][0].anchors
355
356                 inp_dim = int (self.net_info["height"])
357
358
359                 num_classes = int (modules[i]["classes"])
360
361
362                 x = x.data
363                 x = predict_transform(x, inp_dim, anchors,
                        num_classes, CUDA)
364
365                 if type(x) == int:
366                     continue
367
368
369                 if not write:
370                     detections = x
371                     write = 1
372
373                 else:
374                     detections = torch.cat((detections, x), 1)
375
376                 outputs[i] = outputs[i-1]
377
378
379
380         try:
381             return detections
382         except:
383             return 0
384
385
386     def load_weights(self, weightfile):
387
388         #Open the weights file
389         fp = open(weightfile, "rb")
390
```

```
391
392
393
394         header = np.fromfile(fp, dtype = np.int32, count = 5)
395         self.header = torch.from_numpy(header)
396         self.seen = self.header[3]
397
398
399
400         weights = np.fromfile(fp, dtype = np.float32)
401
402         ptr = 0
403         for i in range(len(self.module_list)):
404             module_type = self.blocks[i + 1]["type"]
405
406             if module_type == "convolutional":
407                 model = self.module_list[i]
408                 try:
409                     batch_normalize = int(self.blocks[i+1]["
                            batch_normalize"])
410                 except:
411                     batch_normalize = 0
412
413                 conv = model[0]
414
415                 if (batch_normalize):
416                     bn = model[1]
417
418
419                     num_bn_biases = bn.bias.numel()
420
421
422                     bn_biases = torch.from_numpy(weights[ptr:
                            ptr + num_bn_biases])
423                     ptr += num_bn_biases
424
425                     bn_weights = torch.from_numpy(weights[ptr:
                             ptr + num_bn_biases])
426                     ptr  += num_bn_biases
427
```

```
428                     bn_running_mean = torch.from_numpy(weights
                            [ptr: ptr + num_bn_biases])
429                     ptr  += num_bn_biases
430
431                     bn_running_var = torch.from_numpy(weights[
                            ptr: ptr + num_bn_biases])
432                     ptr  += num_bn_biases
433
434
435                     bn_biases = bn_biases.view_as(bn.bias.data
                            )
436                     bn_weights = bn_weights.view_as(bn.weight.
                            data)
437                     bn_running_mean = bn_running_mean.view_as(
                            bn.running_mean)
438                     bn_running_var = bn_running_var.view_as(bn
                            .running_var)
439
440
441                     bn.bias.data.copy_(bn_biases)
442                     bn.weight.data.copy_(bn_weights)
443                     bn.running_mean.copy_(bn_running_mean)
444                     bn.running_var.copy_(bn_running_var)
445
446                 else:
447
448                     num_biases = conv.bias.numel()
449
450
451                     conv_biases = torch.from_numpy(weights[ptr
                            : ptr + num_biases])
452                     ptr = ptr + num_biases
453
454
455                     conv_biases = conv_biases.view_as(conv.
                            bias.data)
456
457
458                     conv.bias.data.copy_(conv_biases)
459
460
```

```
461
462                 num_weights = conv.weight.numel()
463
464
465                 conv_weights = torch.from_numpy(weights[ptr:
                       ptr+num_weights])
466                 ptr = ptr + num_weights
467
468                 conv_weights = conv_weights.view_as(conv.
                       weight.data)
469                 conv.weight.data.copy_(conv_weights)
470
471     def save_weights(self, savedfile, cutoff = 0):
472
473         if cutoff <= 0:
474             cutoff = len(self.blocks) - 1
475
476         fp = open(savedfile,  wb )
477
478
479         self.header[3] = self.seen
480         header = self.header
481
482         header = header.numpy()
483         header.tofile(fp)
484
485
486         for i in range(len(self.module_list)):
487             module_type = self.blocks[i+1]["type"]
488
489             if (module_type) == "convolutional":
490                 model = self.module_list[i]
491                 try:
492                     batch_normalize = int(self.blocks[i+1]["
                           batch_normalize"])
493                 except:
494                     batch_normalize = 0
495
496                 conv = model[0]
497
498                 if (batch_normalize):
```

```
499                         bn = model[1]
500
501
502
503
504                         cpu(bn.bias.data).numpy().tofile(fp)
505                         cpu(bn.weight.data).numpy().tofile(fp)
506                         cpu(bn.running_mean).numpy().tofile(fp)
507                         cpu(bn.running_var).numpy().tofile(fp)
508
509
510                     else:
511                         cpu(conv.bias.data).numpy().tofile(fp)
512
513
514
515                     cpu(conv.weight.data).numpy().tofile(fp)
```

I.5 CODE FOR DATA PREPROCESSING

```
from __future__ import division

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
import numpy as np
import cv2
import matplotlib.pyplot as plt
from util import count_parameters as count
from util import convert2cpu as cpu
from PIL import Image, ImageDraw


def letterbox_image(img, inp_dim):
        resize image with unchanged aspect ratio using padding

    img_w, img_h = img.shape[1], img.shape[0]
    w, h = inp_dim
    new_w = int(img_w * min(w/img_w, h/img_h))
    new_h = int(img_h * min(w/img_w, h/img_h))
    resized_image = cv2.resize(img, (new_w,new_h),
       interpolation = cv2.INTER_CUBIC)

    canvas = np.full((inp_dim[1], inp_dim[0], 3), 128)

    canvas[(h-new_h)//2:(h-new_h)//2 + new_h,(w-new_w)//2:(w-
       new_w)//2 + new_w,  :] = resized_image

    return canvas



def prep_image(img, inp_dim):
    """
    Prepare image for inputting to the neural network.

    Returns a Variable
```

```
    """

    orig_im = cv2.imread(img)
    dim = orig_im.shape[1], orig_im.shape[0]
    img = (letterbox_image(orig_im, (inp_dim, inp_dim)))
    img_ = img[:,:,::-1].transpose((2,0,1)).copy()
    img_ = torch.from_numpy(img_).float().div(255.0).unsqueeze
       (0)
    return img_, orig_im, dim

def prep_image_pil(img, network_dim):
    orig_im = Image.open(img)
    img = orig_im.convert( RGB )
    dim = img.size
    img = img.resize(network_dim)
    img = torch.ByteTensor(torch.ByteStorage.from_buffer(img.
       tobytes()))
    img = img.view(*network_dim, 3).transpose(0,1).transpose
       (0,2).contiguous()
    img = img.view(1, 3,*network_dim)
    img = img.float().div(255.0)
    return (img, orig_im, dim)

def inp_to_image(inp):
    inp = inp.cpu().squeeze()
    inp = inp*255
    try:
        inp = inp.data.numpy()
    except RuntimeError:
        inp = inp.numpy()
    inp = inp.transpose(1,2,0)

    inp = inp[:,:,::-1]
    return inp
```

I.6 BASE TEMPLATE

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.7.5/css/bulma.min.css">
    <script defer src="https://use.fontawesome.com/releases/v5.3.1/js/all.js"></script>

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.7.5/css/bulma.min.css">
    <script defer src="https://use.fontawesome.com/releases/v5.3.1/js/all.js"></script>
</head>

<body>
    <section class="section">
        <div class="container is-fluid">
            <nav class="navbar" role="navigation" aria-label="main navigation">
                <div class="navbar-brand">
                    <a class="navbar-item" href="https://pjreddie.com/darknet/yolo/?style=centerme">
                        <img src="https://www.efs-auto.com/typo3conf/ext/website_template/Resources/Public/Images/efs_logo_ingolstadt.svg" width="100" height="100">
                    </a>

```

```
25                        <a role="button" class="navbar-burger
                             burger" aria-label="menu" aria-expanded
                             ="false" data-target="
                             navbarBasicExample">
26                          <span aria-hidden="true"></span>
27                          <span aria-hidden="true"></span>
28                          <span aria-hidden="true"></span>
29                        </a>
30                    </div>
31
32
33              </nav>
34              <hr>
35              {% block content %}
36
37              {% endblock %}
38              </div>
39      </section>
40
41  </body>
42
43  </html>
```

I.7 INDEX TEMPLATE

```
{% extends  base.html  %} {% block content %}


<!-- 1st Column -->
<div class="columns">
    <div class="column is-narrow">
        <div class="box" style="width: 500px;">
            <p class="title is-5">Camera - 01</p>
            <hr>
            <img id="bg" width=640px height=360px src="{{
               url_for( video_feed )  }}">
            <hr>
        </div>
    </div>
    <hr>
    <div class="column is-narrow">
        <div class="box" style="width: 500px;">
            <p class="title is-5">Camera - 02</p>
            <hr>
            <img id="bg" width=640px height=360px src="{{
               url_for( video_simulate )  }}">
            <hr>

        </div>
    </div>
    <hr>
    {% endblock %}
```