CRANFIELD UNIVERSITY

AMÉLIE GRENIER

3D PANOPTIC SEGMENTATION WITH UNSUPERVISED CLUSTERING FOR VISUAL PERCEPTION IN AUTONOMOUS DRIVING

SCHOOL OF DEFENCE AND SECURITY
Centre for Electronic Warfare, Information and Cyber

Ph.D
Academic Year: 2020 - 2021

Supervisor: Dr. Lounis Chermak
September 2021

CRANFIELD UNIVERSITY

SCHOOL OF DEFENCE AND SECURITY
Centre for Electronic Warfare, Information and Cyber

Ph.D

Academic Year: 2020 - 2021

AMÉLIE GRENIER

3D Panoptic Segmentation with Unsupervised Clustering for Visual Perception in Autonomous Driving

Supervisor: Dr. Lounis Chermak
September 2021

ABSTRACT

For the past decade, substantial progress has been achieved in the field of visual perception for autonomous driving application thanks notably to the capabilities of deep learning techniques. This work aims to leverage stereovision and explore different methods, in particular unsupervised clustering approaches, to perform 3D panoptic segmentation for navigation purposes.

The main contribution of this work consists in the development, test and validation of a novel framework in which geometric and semantic understanding of the scene are obtained separately at the pixel level. The combination of both for the extracted visual 2D information of the desired class provides a 3D sparse classified point cloud, which is used afterward for instance clustering.

Preliminary tests of the baseline version of the framework for *Vehicle* objects were conducted on urban driving datasets. Results demonstrate for the first time the viability for processing of this type of point cloud from visual data, and reveal improvements areas. Specially, the importance of the boundary F-score in semantic segmentation is highlighted for the first time in this application, with an increase up to 32 percentage point in this study.

Additional contribution was made by applying distribution clustering as well as density-based clustering for instance segmentation in a visual based 3D space representation. Results showed that DBSCAN was well suited for this application. As a result, it was proven that the presented framework can successfully provide genuine 3D profile map representation and localisation of vehicles in a urban environment from 2D visual information only.

Furthermore, the mathematical formalisation of the link between DBSCAN's parameter selection and camera projective geometry was presented as future work and a mean to demystify parameter selection.

Keywords

Panoptic Segmentation; Semantic Segmentation; Instance Clustering; Visual Perception; Autonomous Driving.

ACKNOWLEDGEMENTS

Undertaking a PhD can feel like a lonely journey sometimes, especially when finishing one during the COVID pandemic. But I was not alone in this adventure and I would like to express my gratitude to everyone who helped me along the way.

First and foremost, I would like to thank my supervisor and mentor, Dr Lounis Chermak, for his guidance, his endless patience and his unfailing support through the difficult times. He constantly encouraged me to shift my perspective and built up my motivation and my confidence as a researcher.

For their constructive feedback and for sharing their knowledge, I would also like to thank my review committee, Prof. Mark Richardson, Dr Paul Yoo, Dr Alessio Balleri and Dr Luke Feetham, as well as the director of EWIC, Dr David James.

My sincere thanks to Dr David Nam and Dr Alaa AlZoubi, for their recommendations and contributions leading up to my first publication.

I would like to show my appreciation to Bea Kingdon, Dr Annie Maddison-Warren, and Mandy Smith for their help, advice and support in making things run as smooth as possible.

A special thanks to my friend, Dr Akhil Kallepalli, for all the advice and fruitful discussions. I would also like to thank the rest of the group from EWIC and CSA: Carole, Brandon, Emily, Chris, Alejandro, Axel, Hugo, Leon, Gareth, Sam, Alix, Marco, Eddie, Sarah and Sebastian. Thank you all for making my time at Cranfield University enjoyable and memorable.

Eventually, my warmest thanks to Victoria and Christelle for their support in the final stretch before submission, to my parents Dominique and Pascale for always believing in me, and to my sisters Laura and Raphaëlle for sharing their fighting spirit.

Last but not least, to Thomas, a lot of sacrifices have been made and I am very grateful. This thesis would not have been completed without his support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ADAS	Advanced Driver Assistance Systems
AI	Artificial Intelligence
AP	Average Precision
CNN	Convolutional Neural Network
DARPA	Defense Advanced Research Projects Agency
DBSCAN	Density-Based Spatial Clustering of Application with Noise
DDT	Dynamic Driving Task
DNN	Deep Neural Network
EWIC	centre for Electronic Warfare, Information and Cyber
GMM	Gaussian Mixture Models
IoU	Intersection over Union
JSON	JavaScript Object Notation
MSE	Mean Squared Error
ODD	Operational Design Domain
OEDR	Object and Event Detection and Response
ROI	Region Of Interest
SGD	Stochastic Gradient Descent
SGM	Semi Global Matching

PUBLICATIONS

Conferences

A. Grenier, A. AlZoubi, L. Feetham and D. Nam, **"Towards Scene Understanding Implementing the Stixel World"**, *2018 IEEE British and Irish Conference on Optics and Photonics (BICOP)*, pp.1-4, 2018

1. INTRODUCTION

Transport is a fundamental enabler for mobility, it facilitates the movement of persons and goods over ground, sea, air and even space, empowering trade and exchange for numerous sectors. Obviously, transport systems is a wide domain, and this thesis wishes to focus on the urban automotive field through which majority of people on earth are commuting and travelling on a daily basis. Transportation experienced many disruptions enabling to move faster and further away than ever before. The apparition of the automobile at the beginning of the 1900s constituted a revolution by making this commodity accessible at an individual level. For many, owning a car is a synonym to convenience and to freedom: you can go almost wherever you want, whenever you want.

However, this independence is not yet available for everyone. The reasons are diverse, whether it is linked to age restriction, disability, or not having a driving license for example. The key and common point for a majority of them is that they cannot drive because they cannot operate the vehicle themselves. In addition, the incumbent mobility system lacks efficiency, particularly in urban areas where traffic can be excessively dense despite the endeavour of public transports to reduce it.

Thus, the automotive industry faces today the challenge to continue to enable personal mobility, to improve it and to extend it to those who cannot yet afford it; and to do so at a lower cost. Lower financial cost, lower environmental cost, as well as a lower cost in human lives.

Possible answers to these challenges may have appeared in the last two decades: shared taxi vehicles, car sharing, alternative fuel types and viable electric cars. Paradoxically, the most promising response to both reducing the number of car accidents on the road and enabling everyone to use a car, is to remove the human from the equation: let the car drive itself. As for several previous technology revolutions, research in unmanned and autonomous vehicles emanated from a military context. The Defense Advanced Research Projects Agency (DARPA) prepared the ground and demonstrated in 2007 that developing a car that could drive itself in a relatively manageable scenario was feasible [1]. Many among the best engineers and com-

puter scientists who pioneered this technology by participating in the DARPA competitions later joined Google and its effort to pursue the development of autonomous vehicles in more realistic and complex situations [2].

Ten years later, Advanced Driver Assistance Systems (ADAS) features such as adaptive cruise control, lane keeping or collision warning systems are common in commercially available cars nowadays. In fact, new laws are appearing in certain parts of the world to accommodate testing and deployment of this technology. In parallel, new standards are being defined to accompany and guide the development of this emerging technology. And as fleets of autonomous vehicles are already being made available to a selected handful of users, the reality of autonomous vehicles seems just around the corner. Since 2017, a lot of attention and proportional gigantic investments have been made because of the perspectives and potential benefits that this technology promises to offer. Many auto-makers thus envisaged, and sometimes even promised, to roll it out by 2020.

And yet, as of 2021, there is currently no agreed consensus on how to build this technology. It is also still unclear how to deploy it and create a commercially viable ecosystem for driver-less cars since the technology itself is not mature enough and still far from announced milestones. Indeed, the extensive tests, in simulation and on public roads, have brought to light deficiencies highlighting the fact that the technology is still under development.

There are many challenges to automate the operation of a car in urban or semi-urban environments. Currently, as the infrastructures and the vehicles have been designed for human beings, many of these challenges are related to how best can one reproduce the human's behaviour and capabilities. While driving, the brain is engaged in many tasks:

- seeing and interpreting visual cues in order to understand where the vehicle is at any given time,
- abiding by the driving rules in that portion of the road,
- detecting hazards,
- anticipating other's behaviour and potential actions,

- estimating our own future location.

The brain is capable of executing many of these tasks simultaneously in order to decide the best course of actions, and then acting on it by manoeuvring the vehicle. In other words, the challenge is to reproduce efficiently the perception, analysis, decision making process and the actions of physically driving the vehicle.

The human brain is also capable of adapting one's decision and actions as necessary when new information is perceived. Some abilities are innate such as seeing and interpreting one's surroundings. While some other abilities are first learned by repeated practice through classes, readings and experience, and then demonstrated by passing a driving test. This is the case for handling the vehicle and the set of associated rules. Although the rules are defined to a very basic level in such a way that they can be applied systematically which help in normalising vehicles' interactions and behaviours in order to maximise safety.

Driving involves a lot of common sense, in particular when reaching a scenario for which the rules do not quite fit and a course of action still needs to be decided. As common sense is derived from cultural and social context, which is very difficult for a machine to learn. To this day, this constitutes one of the greatest challenges in autonomous driving, and more generally in artificial intelligence (AI).

Furthermore, to be fully autonomous and avoid the types of crashes deriving from human errors, a self-driving car must not only reproduce the positive aspects of human driving but also supplant the negative aspects in order to converge toward an ideal driving model. However, current self-driving and driver assistance solutions do not yet provide this level of safety, as highlighted by tragic fatalities that have occurred since 2016 [2] with ADAS features being used.

Consequently, questions remain, including the following:

- What is the best strategy to achieve full autonomy: should it be achieved by design or reached gradually by improving ADAS features on current vehicles?
- What is the most appropriate sensor suite to perceive the environment?
- How should the software stack be organised?
- What are the most efficient algorithms for path planning or for perception?

- How much of the algorithm process should be covered by a DNN, a “black box”, preventing the algorithms to be fully explainable?

Some of the hypothesis brought to explain some of the most unbelievable incidents and system failures are linked to environment perception: the surroundings were wrongly perceived and interpreted. This can lead to obstacles not being correctly detected and therefore the system cannot react and behave appropriately. The objective of this thesis is to explore what can be achieved using only visual information as means of perception to provide situational awareness to a self-driving vehicle. Awareness is defined here by an environment representation that is sufficient for navigation purposes.

The framework developed must therefore include geometric as well as semantic information for each of the objects detected in the images. The particularity of the work highlighted in this thesis is to leverage the 3D information. While most of the current work in visual understanding sticks to a 2D object detection problem, this work also leverages classic clustering techniques in a domain where deep learning has become predominant since 2012.

Moreover, the most logical way of capturing an object in a three dimensional space would be to use a ranging sensor such as LiDAR or RADAR which output 3D or 2.5D information. Images on the other hand are restricted to 2D information. However, depth information can be retrieved through stereoscopy and projective geometry which is what this thesis is attempting to exploit and improve. Furthermore, proving the viability of using camera to provide 3D object’s location would constitute an interesting commercial benefit giving the affordability of camera sensors compared to LiDAR.

Several key reasons led to the development of the proposed framework. First, the 2D object detection problem suffers from drawbacks. The first one is that due to its 2D nature while dealing with 3D objects, occlusion and truncating is very common in the images. While efforts are made to deal with these, notably thanks to added notation/labelling in datasets or with the help of neural networks that are very flexible, these efforts remain insufficient as they do not solve the problem on the long run, just expose it.

Indeed, once the objects are identified and must be localised, a depth estimation must be made and the occluding object(s) introduce(s) error in the estimation. Our solution, here, proposes to leverage the stereovision setup to reason in a 3D environment directly. The reasoning being that objects should be self-contained and represented as single entities. This means that, except in case of a collision between two or more objects, they are distinct and separated. A similar reasoning independently applied by NVIDIA led them to process bird's eye views [3].

The second inconvenience of the 2D object problem is that very early, the output was defined as a bounding box with a class probability. This was adequate and satisfactory to hold public challenges, competitions, and to establish benchmarks that have allowed great advances in the field. However, as highlighted by NVIDIA [4], it can be argued that these bounding boxes have become imprecise as they cover a larger area in the image than the actual target. In the field of autonomous driving, bounding boxes are considered more than enough as there is always a margin of error needed to navigate around objects, therefore removing the need for a very precise contour definition in the image of the found object [5].

Although it is true that a margin of error is always necessary for navigation purposes, a bounding box does not enclose objects precisely enough to estimate its 3D localisation from images. Should the object's distance be estimated by taking the minimum depth value within that bounding box? In that case, is that value from the object itself, from the space in the corners or maybe from an occluding object in the foreground? Should an average of the depth values be estimated instead?

Though the latter seems much more satisfactory, it is not; as it incorporates in the estimation of the object's distance many depth values that are not from the said object. Also, it can lead to an estimated distance further away than the foremost point of the object. As a solution, the work in this thesis leverages the pixel-wise semantic segmentation techniques, in order to obtain object points that solely belong to the object and not of their surroundings.

Finally, classification is crucial for navigation. In the autonomous software stack, an understanding of the motion of other objects is important to plan the ego-vehicle's own motion. In order to anticipate other's motion, it is required to know how they

usually move and therefore what they are.

The task that this work aims to achieve is called panoptic segmentation, which is the combination of segmenting images at the pixel level both based on their class and object instances. The hypothesis tested in this work is whether it is conceivable to address 3D panoptic segmentation with traditional unsupervised clustering and without learning an image representation with a deep neural network (DNN) prior to clustering.

With all above considerations, the developed framework is a combination of a depth estimation using stereovision and a semantic segmentation of the left image using DNNs. It is followed by retrieving class-specific points in the depth map to form a 3D points cloud of a given class. Classic clustering techniques are then applied on this sparse points cloud to detect each instance separately.

As the behaviour of the developed framework is explored, this work shows the importance of a good semantic segmentation and highlight the importance of the Boundary F score metric for this application. Indeed, classic clustering techniques are very sensitive to the features they are given and also to outliers. Therefore the better the segmentation, the better the clustering is afterwards.

Two types of clustering that were judged appropriate for the application are investigated: distribution clustering and spatial clustering. The work demonstrates the limitations of distribution clustering on this framework based on visual data. The work also presents the advantages of using a spatial clustering as well as deriving ways forward and the further work it would bring for visual based perception for situational awareness and navigation in urban environment.

The thesis is structured as follows:

1. Chapter 2 gives an overview of background and related work in computer vision and machine learning for robotics' navigation and autonomous vehicles.
2. Chapter 3 describes the research problem and gives an introduction to the developed pipeline and the conducted preliminary tests that laid the foundation of the presented work.
3. Chapter 4 concentrates on the first step of the pipeline, which is the semantic

segmentation, and its influence on the rest of the pipeline.

4. Chapter 5 explores the different clustering techniques and describes their respective behaviour within the framework
5. Chapter 6 incorporates into the framework the improvements discovered in the previous chapters. The resulting version of the framework is tested on a case study for validation and a general discussion.

The domain of autonomous vehicles, both including computer vision and AI, evolves at a very fast pace. So to put the context of this research into perspective, it is important to note that the project started in early 2017 - at the time when self-driving cars were just getting known on the wider public stage.

2. BACKGROUND AND RELATED WORK

The race for self-driving cars has become in the past few years a trending and exciting topic of research. Building an autonomous vehicle is not a straightforward task and requires several domains of expertise to come together: robotics, engineering, computer science, etc. In this chapter, I present briefly the industry to set the context and then describe in more details the perception task in autonomous driving. As this work utilises visual data only, a particular focus is given to computer vision, including a description of projective geometry. Finally, I address the current methods to interpret visual information, centred around deep learning and the potential it has shown on this application in the past years.

2.1 The autonomous vehicle industry

2.1.1 Brief history

In spite of the innovative and sometimes futuristic image that accompanies the autonomous driving world, the idea of a driver-less vehicle is not recent. Tests were conducted throughout the twentieth century, from radio-controlled cars, to detectors and guidance mechanisms within the infrastructure. Around the 90s, more efforts were invested in developing vehicles that could drive semi-autonomously, initiated by the likes of Carnegie Mellon University's Navlab, or Daimler-Benz for example with the Prometheus project [6]. However, the development of fully autonomous vehicles became more widespread and tangible when the American military organised a series of challenges where no human intervention was allowed.

In 2004, DARPA organised the Grand Challenge, a race of 150 miles for driver-less cars taking place in the Mojave desert. None of the fifteen vehicles that entered the race crossed the finish line, and within a few hours - 1/3 of the race - only four vehicles remained operational. The race was considered a failure, although it succeeded in generating interest in autonomous driving technology. The competition was held a second time in 2005 for 132 miles, and this time five vehicles - out of twenty-three - finished the race [2].

For both races, the objective was to follow GPS waypoints, disclosed only two hours

before the start, along a route freed from dynamic objects. As it was a static environment without obstacles other than the rough terrain itself, the main challenge was to control the vehicle and to navigate the road: maintaining its path following tunnels, sharp turns while surviving the dust and going as fast as possible. As a consequence, the competitors did not reach the level of autonomy DARPA had initially hoped for. Eventually, the Urban Challenge was organised in 2007 in order to make the driving scenarios more realistic by introducing traffic, infrastructure and driving rules.

Alongside the race, the event was comprised of three different tasks established to assess the capabilities of the self-driving vehicles. The tasks would not be disclosed before the race start to avoid any pre-programming the drive and encourage the development of some form of intelligence, awareness and route planning. Task A was to turn left on a busy street. Task B was to navigate a street with stationary parked cars on both sides, enter a busy parking lot and pull in and out of a spot without any collision. And finally, task C was a series of four-ways intersections and an unexpected roadblock before the end goal, forcing the robot to stop and reroute itself completely autonomously.

The principal contenders for these DARPA challenges were Stanford University and Carnegie Mellon University, who respectively won in 2005 [7] and in 2007 [1]. The completion of the Urban Challenge was a key milestone in the development of self-driving cars as it showed it was feasible: a vehicle could navigate on its own in a urban environment while managing traffic.

However, industry did not pick up on the momentum of the victory and efforts to pursue the development of self-driving cars came only later, at the end of the decade. Google hired many of the Grand Challenges competitors to gather them under the project Chauffeur. This project would then turn into the now well known company Waymo in 2016 [2].

The Society of Automotive Engineers (SAE) defined in 2014 in the document standard J3016 different levels of autonomy depending on:

- the driver attention and driver's actions required,
- the Dynamic Driving Task (DDT) accomplished by the vehicle,

- the Operational Design Domain (ODD), i.e. conditions, in which the vehicle can operate.

There are 5 levels of autonomy ranging from 0, no autonomy, to 5, full autonomy. It is important to understand in this ranking that the human driver is required to drive from level 0 to 2, as the vehicle is comprised of only assistance features. From level 3 to 5, the vehicle has automated driving features; however the human driver is still required to drive for level 3 if the vehicle requests it.

2.1.2 Who are the actors now?

The landscape of the autonomous driving industry nowadays involves many actors world-wide, and is changing rapidly. The domain is now more than ever built on partnerships between established companies, startups and universities. In this section, I aim to give an overview of actors that have come to light in the past 5 years.

In China, AutoX has been the first to test on public roads and now operates a large and fully autonomous robotaxis service in four cities, including Shanghai and Beijing [8]. Baidu showed in 2017 a driverless solution based on monocular cameras and end-to-end deep learning [9]. While most of the commercially available cars are comprised of level 2 ADAS features, in Japan, Honda became the first to release on the market a level 3 system with a ”traffic jam” assistance feature [10].

In the United Kingdom, Oxbotica was founded in 2014 and developed from a startup into a world leader in autonomous driving software [11]. In parallel, Wayve was founded by a researcher from the University of Cambridge and was the first to test their vehicle, based on end-to-end deep learning, in traffic on UK public roads [12]. Elsewhere in Europe, the likes of BMW, Daimler, Paris Region and Valeo can be cited as competitors in the race of self-driving cars. In Russia, Yandex [13], is particularly impressive for handling adverse weather conditions, snow notably.

The stage of the autonomous industry in America is relatively dense as companies initially from different domains are now working towards self-driving cars at different levels, and sometimes in partnerships: automakers, hardware and software companies, taxis companies, etc. A non-exhaustive list includes:

- Aptiv

- Ford, with Argo and BlueCruise System
- General Motors, with Cruise
- Lyft, now acquired by Toyota
- Mobileye (Intel)
- NVIDIA
- Tesla
- Uber, now acquired by Aurora
- University of Toronto, with Autonomoose
- Waymo (Google)
- Zoox (Amazon)

Mobileye is noticed for having multiple partners and being a solid software developer, supporting three main tasks in autonomous driving: sensing, mapping and driving behaviour. Its solutions are mostly based on cameras although radars and lidars are needed for redundancy. Tesla, known for not relying on lidar technology but on eight cameras providing 360 degrees of visibility [14], was one of these partners until 2016. Tesla is also known for making its “Full Self-Driving Beta” software available to its customers.

For the last decade, taxi companies were particularly interested in the technology in order to deploy robotaxis fleet. The fact that Uber and Lyft have recently disengaged from the development of the autonomous driving technology despite large investments, shows that the viability of this technology is not as straightforward. There are significant debates and development challenges over the business model to adopt. Some are turning towards robot delivery services or to trucking like Aurora. The large amount of funds that are necessary encourage some of these actors to team up, and there is an important turnover in these partnerships.

2.2 Perception of the environment

Tesla and Uber are unfortunately known to cause the first fatalities due to testing this technology on public roads [2]. In 2016, a Tesla Model S hit at 74 mph the middle of a

tractor-trailer that was crossing the road, killing the driver instantly. In 2018, as Uber was testing its autonomous mode, the vehicle hit at 40 mph a pedestrian who was pushing her bag-laden bicycle across the road. For both crashes, the environment has been sensed and interpreted incorrectly by the autonomous vehicle, leading up to these accidents. Whether or not the tractor and the pedestrian have been detected or disregarded as "false positives", both crashes highlight the crucial role that environment perception plays in the autonomous driving technology.

2.2.1 Representing the environment for navigation

The main purpose of mobility is to go from one place to another. In order to do so automatically, the autonomous vehicle must navigate its surroundings, plan a path to the desired destination and then control the vehicle along this path. Navigation necessitates several components to come together and a standard decomposition of the necessary processing in the software stack is the following [15]:

- environment perception
- environment mapping
- motion planning
- control and guidance
- system supervisor

While the system supervisor module checks for hardware faults, software inconsistencies and validates the ODD, the motion planning module determines the autonomous vehicle's behaviour and eventually its path. Both environment perception and mapping modules interact and give an interpretation of the surroundings to the motion planning module. Perception, here, is the process of giving awareness to the autonomous vehicle about:

1. its own *state* in space: position, velocity, acceleration, orientation, angular motion.
2. the identification and localisation of other elements in the surroundings.

The second, Object and Event Detection and Response (OEDR), is a crucial criterion

for automation as it identifies what affects the driving task and how to react to the surroundings appropriately. In order to make *informed* decisions the vehicle needs to be aware of two categories of elements: *static* and *dynamic* elements.

The former accounts for elements that generally belong to the infrastructure, such as the road, curbs and markings, traffic signs, construction areas, buildings, etc. Identification is here important for the vehicle to adapt correctly its attitude and driving for the safety in construction sites, school areas, and respecting the driving rules in general.

The latter includes other vehicles, pedestrians and other moving obstacles to avoid. Identification plays a critical role to distinguish entities in order to anticipate their movements and trajectory correctly for the vehicle's motion planning.

Different techniques exist to represent the environment and visualise the different elements mentioned above. To avoid collision, static elements can be represented with occupancy grids [16], where each cell in the grid covering the surroundings is determined by its likelihood to be occupied. Representations typically involve reproducing the 3D scene as a point cloud or summarising the information about the surroundings in a more compact manner: voxels, stixels, meshes etc [17] [18].

Some detailed road maps can also include precise positions for all regulatory elements, and lane markings for mission planning. Some are created "offline", before driving, and are pre-driven in simulation. Others are created "online" by populating the map according to the sensors inputs. In any case, both modules of environment perception and mapping always interact in order to update and create the most accurate map so that the local navigation, i.e. immediate actions, can be made safely.

Note however, that the decomposition of the software stack into several manageable blocks is not always the strategy employed by companies to develop autonomous vehicles. Some utilise end-to-end learning [19] [20] which consists in using deep learning techniques as replacement of the software stack entirely. This is usually done with a Deep Neural Network (DNN) that outputs the required control actions with only the sensors measurements as inputs. These approaches are made possible with the progress achieved this past decade in hardware and DNN development.

2.2.2 Sensing the environment

These maps of static and dynamic objects used to plan a path for the autonomous vehicle to follow are populated using sensor measurements. All companies mentioned in Section 2.1.2 use a combination of all or some of the following sensors: GPS, IMU for internal state measurements, and radar, lidar, ultrasonic or visual sensor for external measurements.

Whether the external measurements are used to estimate the ego-vehicle's state or to determine the content of the surroundings, the perception task faces many challenges. Among these challenges, coping with data uncertainty and data loss are important in order to enable robust and consistent interpretation of the sensor data.

Uncertainty and loss can come from various sources: occlusion and reflection bringing ambiguous information, illumination for cameras, lens flares, adverse weather conditions or terrains. Therefore, sensor fusion, which leverages redundant information from multiple sources to address these challenges, is crucial.

However, there is no consensus on which is the best sensors suite to implement despite conducted studies [21] and theoretical contribution of each sensor type [22].

Since the development by Hall of the Velodyne lidar in the year 2000s, a 360 laser range finder is a key component of environment perception and mapping as it gives a 3D point cloud of surrounding obstacles several times per seconds. Active sensing is usually more accurate than passive sensing and also requires less processing. However, despite efforts in reducing the price of LiDAR technology, this is still a very expensive sensor in comparison to cameras. Also, it is argued that when combined with radar, lidar does not contribute enough for its cost.

There are several arguments towards the usage of cameras for autonomous driving perception alongside its relative cheap cost [22]. Indeed, in clear weather, they achieve the best range for obstacle detection, although images suffer from imprecise depth estimation in comparison to lidar or radar measurements. However, images also provide more information than lidar or radar that provide basic geometric knowledge, and to a lesser extent, material knowledge of the scene.

Notably, the roads have been designed for the human drivers, whose eyes are the

main vector of information input. Consequently, images and colour information are necessary to read signs and lane markings or to detect another car's blinker for example. With this fact only, cameras are the sensor type that is closer to human eyes perception and so is essential to achieve autonomy in the current state of the world.

Therefore, using visual sensor does not only make sense in term of perception but has additionally a real cost-saving incentive. Visual odometry is already an established domain to estimate the ego-vehicle's state and localisation based on vision. With the emergence of DNN, object detection and recognition on images has rapidly evolved in the last decade [23].

2.3 Working with images

To reconstruct a 3D scene from a 2D image, several key components are needed: the calibration of the cameras and an estimation of the scale, which is, here, a depth estimation using stereopsis. A brief description of the underlying principles are outlined in the next sections.

2.3.1 Mathematical model and calibration

The pinhole camera, or camera obscura, is a box with a small aperture that can be assimilated to a point and by which the light enters into the box. The light rays then reach and hit the back of the box, also called image plane, to form an image. In practice, this camera is never used because it does not collect enough light to form a clear image.

However, the setup serves to derive from the light rays trajectory the basic mathematical projection formulas which are needed to reconstruct a 3D scene from an image. The pinhole camera is supposed ideal: the image plane is correctly aligned with respect to the pinhole, its optical centre.

Therefore, the pinhole camera model represents a simple camera without the lens, and with a single aperture as small as a point instead. Considering the model in Figure 2.1, the focal length f of the camera and similar triangles, the projection of a 3D point $P(X,Y,Z)^T$ into the image plane at a pixel position $(u,v)^T$ can be written as:

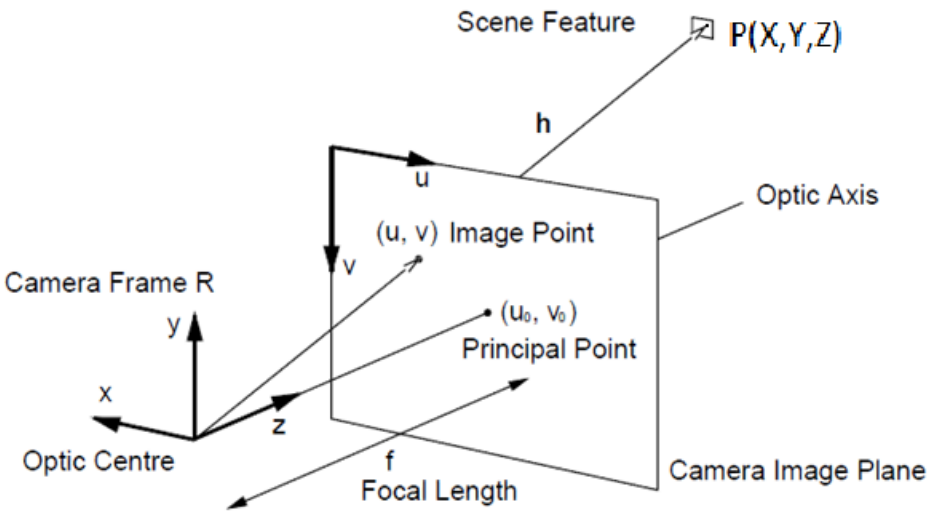


Figure 2.1: Geometric model of the pinhole camera

$$u = f\frac{X}{Z} \quad \text{and} \quad v = f\frac{Y}{Z} \tag{2.1}$$

If u and v are considered to be the size of the sensor used, this relation can be used to determine the focal length needed to see an object of height Y at a distance Z for instance. The above expression can be written in a matrix notation if a scale factor w is introduced.

Indeed, one can observe from the pinhole camera model that a projected point in the image can, mathematically, come from a multitude of 3D points along the light ray. To have a unique solution to the projection, an estimation of the depth of a specific 3D point along the Z axis is needed; i.e. its distance from the camera optical centre. Thus, in Equation (2.1), the scale factor is Z.

In reality, the manufactured cameras are not guaranteed to have the optical centre perfectly aligned with the centre of the photosensitive sensor that measures the intensity of the received light. Some other parameters such as the skew factor s and the principal point (u_0, v_0) are introduced to capture these offsets and errors that are unavoidable during production. Calibration therefore needs to be performed [24] to estimate the intrinsic camera parameters, which represents these specific properties that are unique to each camera. By convention, this matrix is noted K.

It is important to note here that it is impossible to obtain the exact and absolute value for f in meters, nor the size and shape of the pixels without dismantling the camera. The calibration instead gives an approximate of the focal length in pixel in each direction of the image, (f_x, f_y).

The camera model also includes the extrinsic parameters of the cameras which represent the pose of the camera in the world coordinate system. In other words, this is a description of where the camera is placed within the system. Therefore, this transformation matrix contains a rotation matrix R and a translation vector t.

The final equation to project a point from the world coordinate system to the image plane is the following:

$$w \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} R & t \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.2}$$

Because the small aperture of the pinhole camera does not allow to receive enough light, an optical lens is placed in front of it to gather more light in practice. To the above calibration, the distortion coefficients of the lens must be added. It is also important to use a camera without an adaptive focal length, otherwise the calibration process will be rendered void.

2.3.2 Stereovision and disparity maps

Though many techniques nowadays estimate the depth from a single camera [25], initially, two images of the same scene from different perspectives are needed to obtain a perception in 3D. This reproduces the human binocular vision, and is illustrated in Figure 2.2.

For the same feature, which is a specific, unique and recognisable point in the image like the smiley face in Figure 2.2, the position difference in pixels between the left and right images is called the disparity and is noted d. Using Equation (2.1) derived from the pinhole camera model, the distance Z of this feature point in the scene from the camera can be linked to the disparity by Equation (2.3), where T is the distance

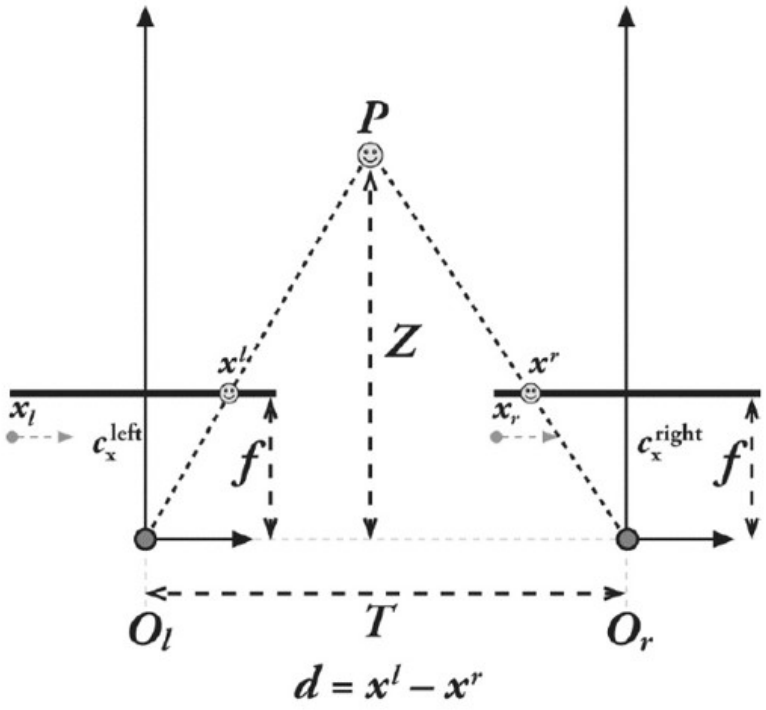


Figure 2.2: Geometric model of a pair of cameras, from [26]

between the optical centres of the left and right cameras.

$$Z = \frac{f * T}{d} \tag{2.3}$$

In order to estimate distance for the entire image, features are detected in the left and right images in order to build a disparity map. It is essential here that the pair of images are rectified so that the optical axis of the two cameras are parallel. This enables to use the epipolar line constraint of the stereovision setup and thus to find each corresponding feature on the same row in both images.

There are several techniques to detect and describe features [27] [28] [29] [30], but all features must be distinct, repeatable, local, numerous and found within a reasonable computation time.

These features are then paired up using feature matching algorithms, brute force matching or otherwise. Using the paired features, the disparity can be estimated and the distance calculated with Equation (2.3).

Despite efforts and development made in the last few years, stereovision suffers from a quadratic error, increasing with distance. This means that the technique is useful at short to mid-range but adjustments must be made, or other sensors used, to perceive at further distance. This drawback directly impacts the work in this thesis and explains both the scattering that will be observed in the projections as well as the restriction of the case studies to the immediate surroundings of the vehicle.

2.4 Interpreting visual information

There are many aspects in image processing that can make interpreting visual information challenging such as:

1. Viewpoint variation in images
2. Change of illumination
3. Deformation
4. Occlusion
5. Background clutter

In recent years, deep learning applied in computer vision has shown promises on these challenges compared to traditional approaches. In computer vision, object classification, object detection, depth estimation, and semantic segmentation are four different tasks. Each has an image as input but the expected output changes and is respectively: a class label, a class label and a position in pixels, depth for every pixel, a class label for every pixel.

For object classification, the ImageNet challenge is a reference and each year allows for the state-of-the-art to be ranked. Since 2012, the top methods are most entirely based on neural networks, the first one being AlexNet [31]. The latter is an adaptation from the first convolutional neural networks created by LeCun, LeNet-5 [32] for a GPU implementation immensely accelerating the computation competitiveness of the CNN.

The CNN are particularly appropriate to work with images, and it has been shown that the first layers of the feature detectors reproduced, by learning on its own, similar features detectors that already existed in image processing. [33] This parallel can

be made thanks to the cross-correlation that each neuron in the layers implements. Each neuron is based on the perceptron that consists of a set of weights, which in a CNN acts as a filter kernel, biases and an activation function. Activation functions depend on the task, the layers and can be one of the following: Sigmoid/logistic, softmax, tanh, and ReLu. However, “Regular neural nets don’t scale well to full images.” “This full connectivity is wasteful and the huge number of parameter would lead to overfitting.” [34]

Neural networks architectures change and evolve depending on the problem the learning algorithm is trying to solve. A non exhaustive list of DNN architectures includes Yolo [35] [36], Inception [37], R-CNN [38] [39], VGG [40], SegNet [41], DeepLap [42], EfficientPS, etc. Object detectors with sliding windows became efficient thanks to Overfeat [43], which saved a lot of the computation by recognising that the algorithm repeated many of the same operation and computation.

Object detectors based on bounding boxes often have a “Non-max suppression” algorithm to filter their neural networks outputs, keeping the most probable outputs (with a high confidence score). There is a transition from bounding box detections towards “blob” segmentation. When the metrics were established for a boundary score in segmentation task, the authors suggested this kind of precision for task like driving would not matter as much as for graphic tasks. Today, we see a shift in that trend as NVIDIA also works on segmentation, not only on bounding boxes.

Before being used, these neural networks are trained on labelled data, as this is supervised learning. The training process is an optimisation problem, where the cost of a loss function is minimised. An example of loss function is the cross-entropy loss. The first and commonly used optimisation algorithm is gradient descent. Its implementation for networks, the backpropagation, originates from Hinton’s team in 1986. Variants and other algorithms have appeared in the last decade: SGD, SGD with momentum, etc. Due to the complexity of the networks, some methods of regularisation are developed to avoid overfitting: L1 regularisation, and the most popular today: Dropout.

Neural networks are trained with cross-entropy loss, which is linked to the Shannon’s entropy. Neural networks are tuned to extract information, and once trained,

it does it extremely efficiently thanks to architectures that are essentially matrices multiplication. However, the training and the amount of data required are gigantic.

Applications of neural are not limited to autonomous driving nor image analysis. However, the enthusiasm for deep learning is such that it is tried in wide range of applications. According to Bengio et al [44], a neural network has the possibility to learn any complex function, which makes it a very powerful tool, assuming that there are enough resources to train it.

This type of neural networks is particularly well adapted for images thanks to the convolutions embedded in the network. In computer vision and image processing, convolutions have long been used to pass images through filters. A filter is usually composed of a kernel, a matrix of a given size (3, 5 or more pixels), and is combined to the image following this expression:

The result is a third image that shows how the original image has been affected by the shape of the filter (determined by the kernel values). It can be used to extract or sharpen edges, reduce or create noise, etc. It has been shown in [33] that in a CNN, the first layers of filters learned by the network are actually similar to common filters used in image processing, before moving on to learning more complex filters in deeper layers. In the CNN, the output of these filters at every layer are features, specific characteristics of the image that are extracted and then used to identify the image content, as shown in the figure below.

In this thesis, basic CNN approaches are not used for several reasons. First, image rendering of urban traffic scene mostly consist of several objects to be recognised, and basic CNNs tend to give one class label per input image. Second, CNNs that are capable of multiple classes recognition do not necessarily give information on where in the image the objects are. Indeed, CNN are invariant to translation. This means that, as the filters kernels are passed through the image and thereafter through the deeper layers, the necessary features for recognition are extracted wherever they are in the image: a cat will be recognised whether it is on the left, the right, the bottom or the top of the image.

This renders CNNs very robust and flexible, however, the location of the feature, and thus the object, is not directly saved. Modern object detectors with CNNs will use

sliding windows (R-CNN and other versions of it) or anchor boxes (Yolo) to determine the position of the objects in the image. These detectors stay within the definition of the 2D object detection problem and are not suitable.

Since AlexNet [31] won the ImageNet Challenge [45] in 2012, Computational neural networks (CNN) have been a strong component of object classification and detection in Computer Vision. Although neural networks are loosely inspired from neurons in the brain, they are mathematical tools used to approximate complex models and so, they do not learn like humans do. They do not think. Millions of examples can be shown to them but they will recognise what they have been given to see. Indeed, they are easily tricked by edge cases because generalisation and conceptualisation are not trivial. A leaf in front of a sign or an unusual orange cone are banal examples for human drivers that can cause the neural networks to fail.

As part of the supervised learning category of machine learning techniques, they require a training process before being used. The training consists in giving the network labelled examples so that it learns to adapt its output to what is expected: the given label. A network is composed of several units, also called artificial neurons, each of them having trainable parameters called weights and biases.

The training, i.e. the learning process, will tweak and change these parameters until the network gives the correct answer to the given examples, hence the name of supervised learning. The training requires resources like data, computation power and time. However, once trained, the network is more efficiently used.

The unit is based on the perceptron it takes some inputs, does a weighted sum of them and passes it through a non-linear function called activation function. If the input of the unit is considered important for the task, the weights will be adjusted so that the sum activates the neuron, i.e. passes the threshold of the activation function, and is passed to the next neuron.

The perceptron alone is not sufficient and needs to be combined with other neurons in order to form a more complex function and be capable of solving more complicated problems. The architecture of the networks varies depending on the task it performs. For object detection in images, the CNN architecture proved to be efficient. The unit still follows the principle of the above-mentioned perceptron but is adapted to fit

image processing by replacing the weighted sum with the correlation operator.

An example of CNN architecture used for object classification is the VGG 16 [40]: convolutional layers alternate with pooling layers to form the feature extractor, which is then connected to fully connected layers to classify the image based on the given features.

Another type of approach is called panoptic segmentation which consists of performing semantic segmentation and instance segmentation simultaneously. If semantic segmentation relates to the classification of individual pixel according to pre-defined classes, instance segmentation on the other hand, refers to the grouping of pixels into instances. Instance segmentation can be divided in two categories for images: the ones that creates a bounding boxes and then refine the boundaries mask (derived most of the time from object detection), and the ones that "color" instances, labeling pixels directly.

Instance segmentation is a complex task which main challenge lies in figuring out the adequate number of the sought instances. This information can be dealt either in 2D using visual based approaches [46] [47] [48] [49] [50] [51] [52] [53] [54] [55], or 3D using 3D point clouds based on lidar measurements [56] [57].

2.5 Summary

The autonomous driving development was propelled by key actors in both academia and industry, but also by specific set of challenges and established benchmarks to push the algorithms further. This has led to many ADAS features for modern vehicles, some that are already available and/or considered as safety standards. Some other features are, on the contrary, still being tested.

There is an important debate over the strategy required to achieve full autonomy, referred to as Level 5. There is not yet a approach performing remarkably better than another, nor is there an established business model for this new technology. Therefore, autonomous navigation is still individually shaped by each actor and the end application.

Technologically, building a self-driving vehicle means finding a balance between sensor suite, algorithms, data resources, computational time and energy usage. This has proven to be a difficult challenge as a technology consensus has not yet been reached.

For safe autonomous navigation in short to mid range, local planning needs a detailed map and representation of the immediate surroundings of the ego-vehicle in order to perform obstacle avoidance. It also requires the detection and localisation of the other dynamic objects in the “3D real world” scene with respect to the ego-vehicle.

In order to perceive the environment using visual data, there are a certain number of tasks that are required including, but not limited to, depth estimation, object detection and recognition. These tasks consist of separate entities of research with their respective benchmarks, established definitions and evaluations metrics. Nowadays, these computer vision functions can be computed using techniques most entirely based on deep learning, which are challenging to interpret with clarity.

3. RESEARCH PROBLEM

This chapter presents the research problem and the related contributions proposed as a possible answer. First, the questions on which this work focuses are stated, as well as the main objectives. Second, a description of the framework I developed during this thesis is given. Finally, the proposed framework in its basic functional form is tested. Evaluation metrics and the data generated in support of these preliminary tests are outlined.

These preliminary tests act as a proof of concept. They aim to demonstrate the viability of the implemented framework and highlight limitations and improvement areas. The latter will be investigated in detail in Chapter 4 and Chapter 5 before concluding with a final case study in Chapter 6 aiming to validate the framework with the proposed improvements

3.1 Question and objectives

This work aims to derive from visual information the necessary characteristics in 3D of dynamic objects present in the scene to provide for the navigation module. In a self-driving car context, perception and understanding of the ego-vehicle's immediate surrounding is critical. Indeed, to plan its own motion, the ego-vehicle needs to localise itself in the environment as well as the other dynamic objects and predict their behaviour.

Indeed, in order to plan a path for itself that would be free from collision with other agents on the road, it requires the knowledge of these agents' locations. This is achieved through detection and tracking of these agents allowing to anticipate for their motion in the near future. Consequently, the essential characteristics about the other dynamic objects that are required for a safe navigation include:

- classification
- current position, in the world coordinate frame or relative to the ego-vehicle
- heading
- velocity

It could be argued that it is not needed to understand what an object is to avoid it; however, knowing what it is actually greatly helps motion prediction models. Indeed, a bus, a car, a bicycle and a pedestrian have all different behaviours, different movement constraints and different capabilities that make their potential path different from one another.

For examples, a bus takes larger curves to accommodate for their length, the heavier the vehicle the more inertia they exhibit, pedestrians have a relatively low speed but can show more rapid changes in direction, etc.

This deterministic behaviour is also restricted by the driving rules, downsizing the number of possible paths other vehicles can likely take. For example, a car driving in a lane is likely to keep driving in the lane, abiding by the speed limit. And so optionally and if available, the motion prediction modules can benefit from high definition road maps containing lane boundaries, speed limits and other information of interest.

The history of the objects' states and images of the objects for visual cues such as an activated blinker could also be used to enhance the precision of the behaviour and motion planner.

Among the necessary characteristics though, heading and velocity can be estimated from two, or more, consecutive position estimates. I therefore aim to identify and estimate the current position of the other objects, focusing the problem on ***where*** and ***what*** these objects are.

Furthermore, particular attention is given to the immediate surroundings of the ego-vehicle. This work is limited to the field of view covered by the images in front of the car and the primary interest are the vehicles delimiting the free space ahead of the ego-vehicle. The reason is that for navigation purposes, these dynamic objects in the immediate surroundings are the most important to detect correctly as they are the ones that have the highest probability to impact the ego-vehicle path planning.

Indeed, when following a vehicle in a lane, the car just in front of the ego-vehicle is the one that limits its speed so that vehicles do not crash. Vehicles in the traffic further up ahead on the other hand, while nice to have to anticipate behaviour motions, are not the priority.

Obviously, these priorities shift depending on the driving scenario and driving task, whether the autonomous vehicle is at an intersection, lane keeping, or overtaking. In this thesis, the order of priority listed below is followed as a general rule:

1. the vehicle immediately in front of the ego-vehicle in the same lane
2. vehicles potentially parked on the side of the ego-vehicle's lane, as their status can changed from static to dynamic instantly and they can enter the lane at a moment's notice
3. closest vehicle on the opposite lane

Following the discussion in Chapter 2, autonomous driving has been a continuously and fast pace evolving domain in the past few years and many techniques have been developed recently for scene understanding. The majority of them are based on multiple sensors, including LiDAR, cameras, ultrasonic sensors, radar providing huge amount of data. Interpreting this data usually necessitates deep learning approaches, which require extensive computing resources. Additionally, for techniques solely based on vision, the majority of the developed algorithms for object detection follow a 2D problem definition as locations of detected objects are given in image coordinates, i.e. in pixels.

The objective of this thesis is to take advantage of the camera as an affordable sensor, providing very rich information content, to combine classical computer vision techniques with unsupervised machine learning approaches. This provides an interesting solution with regards to 3D reasoning for panoptic segmentation leveraging depth information in a different manner compared to the presented approaches in the literature. Indeed, depth information is used either like a mere lookup table or as input to a neural network. The latter is an opaque process due to the "black box" nature of neural networks, making interpretation and problem adjustment much more complex. The goal is also to better understand the implications of using only cameras in scene understanding and representation, from acquisition to surroundings mapping.

To summarise the principal objectives of this thesis is to develop a visual based perception framework for navigation purposes that satisfies the following characteristics:

- uses visual data from cameras only,
- detects and classifies the ego-vehicle's immediate surroundings,
- gives 3D position of other vehicles,
- utilises depth estimation differently,
- leverages other techniques than deep neural networks in order to demystify parameter selection,
- combines and leverages existing benchmarks.

The main question addressed in this thesis is whether a framework with the above characteristics is viable. However, the question is not just restrained to the viability of the approach; rather I aim to highlight the new perspective associated with this application as well as the current limitations and necessary conditions to work with visual data only. This hypothesis is tested on vehicle detection, through experiments in specific conditions: limited in computational resources and to a urban scenario in clear weather conditions.

3.2 A modular pipeline

Situational awareness implies having a multi-level understanding of the ego-vehicle's surroundings. Mathematically, from 2D images to a 3D map, there is a space dimension change, that is made possible thanks to sensor calibration and projective geometry. To interpret the scene, processing algorithms search for mainly two types of information: content identification and geometrical structure.

Content identification aims at recognising the nature of what is encountered around the ego-vehicle. Geometrical structure, however, relates to estimating its location and its size. The combination of both geometrical and identification form information at the object level. This information can then be condensed into a detailed map or a simplified environment representation.

The order in which these types of information are extracted may vary depending on the techniques employed. In a previous research I conducted [58], I explored the usage of the compact stixels representation in order to detect the closest obstacles, delimiting the free space around the ego-vehicle.

Stixels represent only the first objects around the ego-vehicle with horizontally stacked pixel sticks, as shown in Figure 3.1. They appear as a flat 2D barrier above a planar ground with the object relative height but a fixed width.

The idea, also researched by others, was to detect obstacles based on the geometrical information to then identify them and produce semantic stixels [59] [60] or instance stixels [61].

Figure 3.1: Example of the Stixels representation from [58], based on the work of [62] and [63]

Instead of identifying obstacles that were previously detected based on geometrical features, the particularity of the proposed framework is to associate identification and scene structure as complementary processes that are combined in order to obtain a 3D objects proposal. In addition, compared to other similar approaches that remain in the 2D space, the object proposal is acquired within a 3D map.

To do so, stereo cameras are used, and a streamline process is implemented using both 3D computer vision and AI tools from deep learning to statistical methods.

From analysing 2D images, this pipeline provides 3D localisation and profile of each vehicle entities from a short to mid-range distance (approximately 2 m - 40 m).

The developed framework takes as input a pair of colour (RGB) images of the scene looking forward, identifies the image content, estimates the geometry of the scene, and combine the information to detect and extract the different objects. Finally, the pipeline provides these objects' locations in 3D relative to the ego-vehicle. This pipeline is depicted in Figure 3.2 and is comprised of five steps, which are:

1. pixel-wise semantic segmentation,
2. depth estimation,
3. filtering and point cloud creation,
4. instance clustering,
5. mapping.

The framework is designed to work as a modular pipeline to leverage state of the art techniques currently available in existing benchmarks. Indeed, steps 1 and 2 are standard tasks in computer vision and benefit from established benchmarks [64] [65] [66] [67] to facilitate advancement in algorithms development. However, step 5 is heavily dependent on the cameras calibration and thus based on the used dataset's hardware. In the following sections, each step is described in more detail.

As a reminder, it is important to note that the simultaneous recognition and detection of objects in images is known as instance segmentation. The simultaneous semantic segmentation and instance segmentation has recently been called ***panoptic segmentation*** [68]. This work therefore develops a framework for 3D panoptic segmentation.

3.2.1 Step 1 - Classifying the content of the scene

First, the left RGB image is segmented semantically pixel by pixel, also called pixel-wise. In other word, a meaning is given to each pixel in the image by classifying them into one of the categories defined by autonomous driving benchmarks. The Cityscapes dataset [65] for example, defined a series of classes and categories that are present in images of traffic scenes: *sky, building, road, vehicles, persons, etc*.

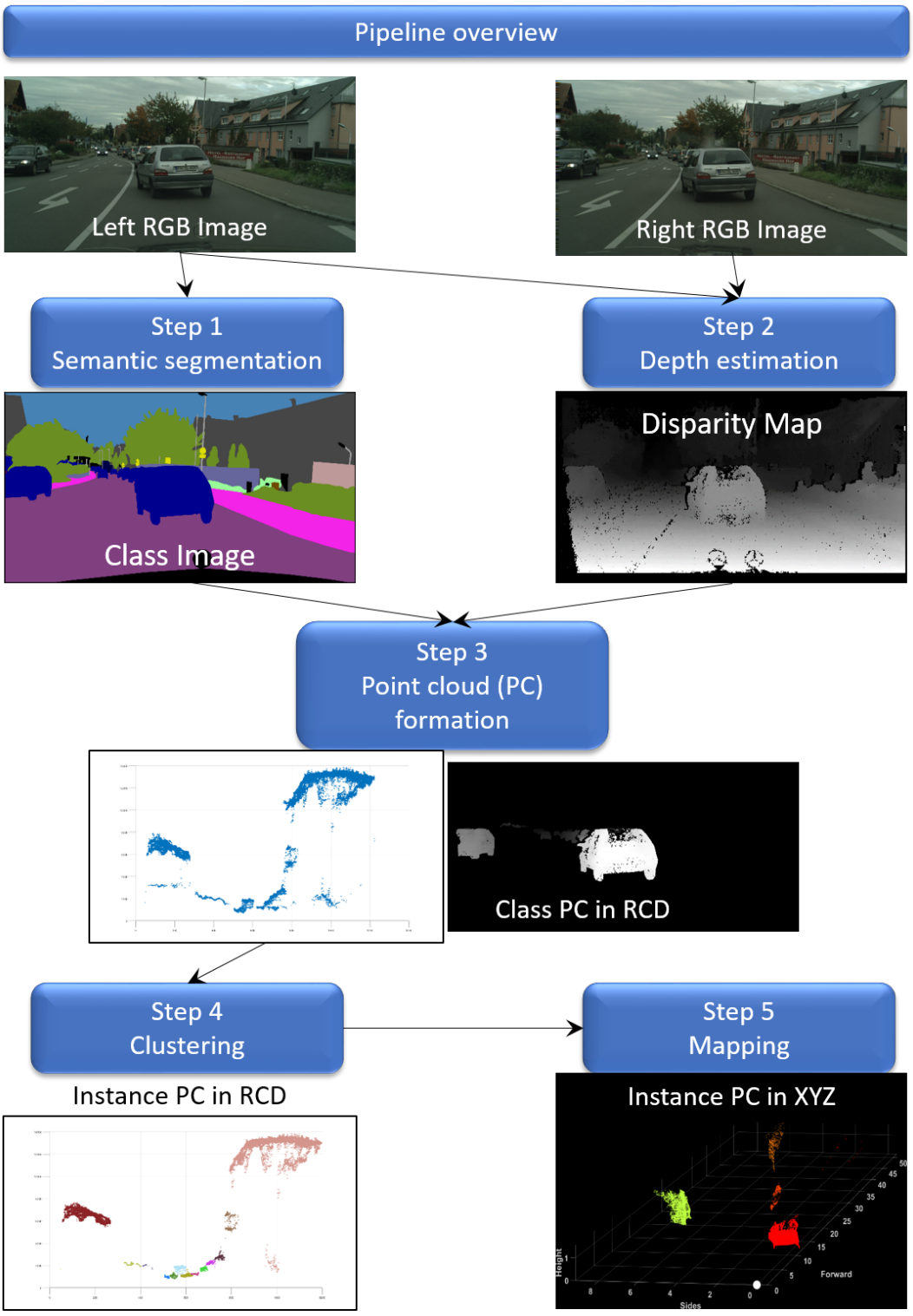


Figure 3.2: Overview of the developed pipeline.

This pixel-wise classification can be performed with encoder-decoder network architectures. Encoders are usually comprised of a CNN for features extraction as described in Section 2.4. As convolutions are performed, the dimensions are reduced from the image resolution to a much smaller size for the feature maps. The decoder thus contains deconvolutions, which allow to recover from this size reduction back to the image resolution size and classify each pixel.

An example of the output of this step is shown in Figure 3.2 as the ***class image***. In Chapter 4, the semantic segmentation is studied more extensively, including the network architecture and its training as well as the step's influence on the pipeline.

3.2.2 Step 2 - Retrieving the sense of depth in the scene

As a class is associated to each pixel in the previous step, this step associates a depth value, extracted from the disparity map, for each pixel in the image. The left and right images are then processed to understand the structure of the scene and obtain an estimate of distance from the viewpoint.

The underlying principle behind this reconstruction is stereopsis, and can be simply illustrated by the way human beings innately perceive depth thanks to binocular vision. By looking at a specific point in the scene from both eyes, the human can triangulate its position and estimate how far it is from the viewpoint. In order to reproduce this depth perception, there are two prerequisites:

- The process of identifying and matching these specific points that are visible on both left and right images is called feature matching. The position of these features in both images slightly differ due to the difference in the cameras' viewpoints. This difference of position noticed by comparing the features' positions in both images is called disparity and is measured in pixels.

- Knowing the location of both eyes, or cameras, relative to each other to use projective geometry. This is done through camera calibration.

This step's output is the ***disparity map*** shown in Figure 3.2, where each pixel takes its disparity value.

3.2.3 Step 3 - Filtering and point cloud creation

Each pixel in the left image is at this stage associated with a class and a disparity value, which eventually corresponds to a depth value. Plotting this information yields a point cloud, similar to a LiDAR point cloud, except that it adds classification to the distance information. Notably, the point cloud density will be obviously smaller than LiDAR's as this is limited to the camera spatial resolution.

In itself, the creation from visual data of a classified point cloud of the scene forward for processing is innovative. It is a step further than RGB-D (Red Green Blue - Depth) images, or than using disparity maps as lookup tables after the objects detection task to retrieve distance information.

As a consequence of the two preceding steps, both the semantic and geometrical structure of the scene are reconstructed. While it is now known where and what each of these points are, it is not known yet which points among them form one single entity. This step aims to find which points belong together to form one instance. Clustering algorithms will be used to do so. These algorithms are in general relatively sensitive to outliers in the data so invalid points need to be eliminated.

One of the complexity of the application resides in the diversity of the encountered situations and the various nature and type of encountered elements. In this filtering step, the complexity is reduced by addressing a single class per point cloud. Consequently to the objectives elaborated in Section 3.1, the points belonging to the *vehicle* class are selected to be isolated and create a vehicle point cloud. Note that the same process would apply for the other classes of dynamic objects that need to be segmented (persons, bicycles).

Following the segmentation step, a "class" image is obtained with the pixel colour defining the object class. From this segmented class image, a binary mask of the desired class is extracted by isolating the pixels whose colour intensity corresponds to the desired object class. This mask is eroded with a square kernel of size 5. This morphological operation serves two purposes.

The first motive is to remove outlying pixels around the segmented areas of the desired class as boundary regions are likely to include misclassified pixels. Indeed, single isolated pixels are likely to be noise from an erroneous segmentation rather

than from a detected entity. The second is to mitigate the effect of poor contours in the segmentation, as wrong contours will incorrectly include or remove points in the resulting point cloud. This effect of the segmentation is explored further in Chapter 4.

It is also important to remember that the disparity map results from a combination of the left and the right images. As the point of views differ, the contours of one object slightly shift in pixel position. Thus, even with decent contours in the segmentation, the overlap between the segmented image and the disparity map is not perfect. In such a case, the morphological operation limits the inclusion of erroneous disparity values that would correspond to the sides of the object rather than the object itself. This is illustrated in Figure 3.3 by the vehicle in dark blue on the right. These inclusions are limited but not prevented completely as the accuracy of the disparity maps could be further improved, as well as the erosion brought by the morphological filter.

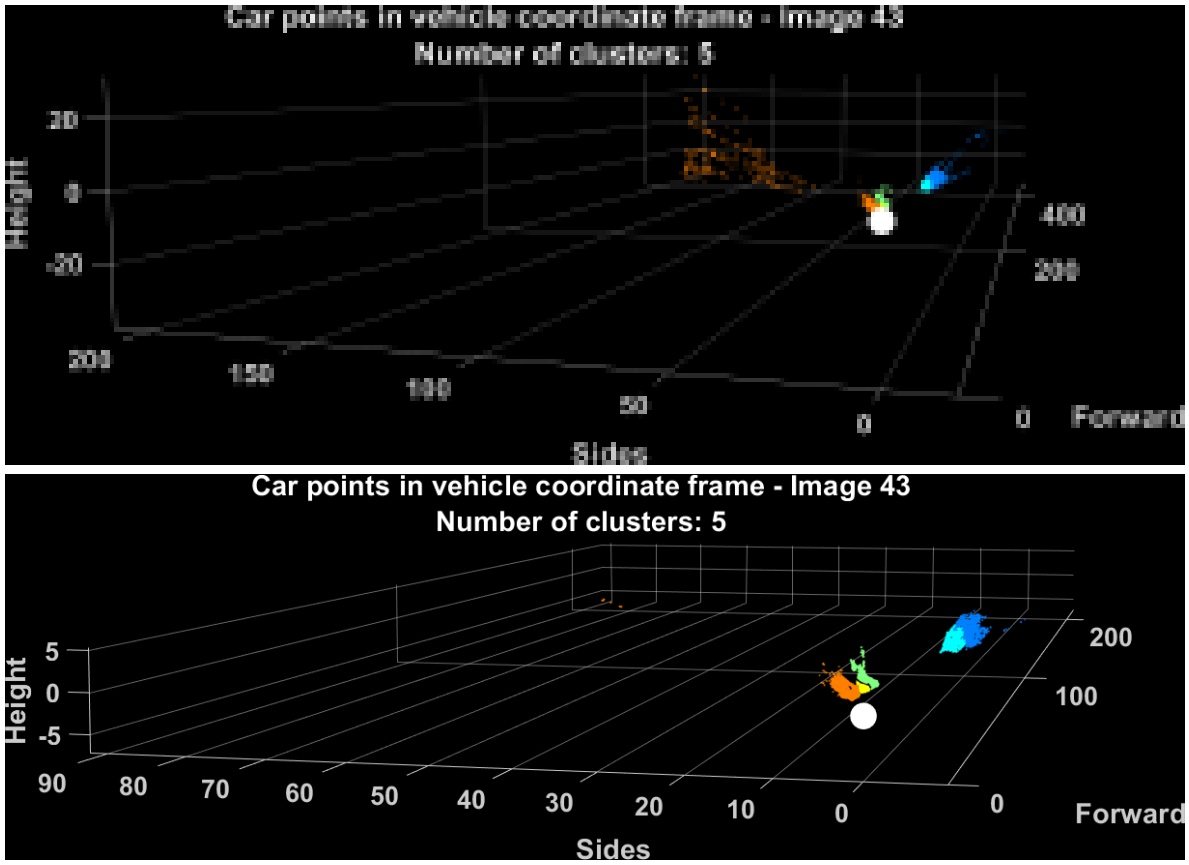


Figure 3.3: Effect of the filtering step on the vehicle point cloud. Not filtered above, filtered below.

Furthermore, the left and right image do not overlap completely, by definition of a stereovision setup. Therefore, many points on the left of the left image will not appear in the right image and therefore result in invalid disparity values in the disparity map. When the segmented image of the desired class contains pixels in this area, the points with invalid disparity values are filtered out. This is illustrated in Figure 3.3 by the erroneous points in orange that are sparsely scattered on the left.

Note that in Figure 3.3, the point cloud is projected in the ego-vehicle coordinate frame to ease visual interpretation. An example of this step's output is shown in Figure 3.2 under step 3, as an image on the right and as the created point cloud on the left. The x axis are the columns of the image, the y axis the disparity values and the z axis (not apparent) are the rows. Recall, the higher the disparity values, the closest the points.

3.2.4 Step 4 - Clustering each object of the scene

At this stage, outliers in the vehicle point cloud have been removed so in this step clustering techniques are used to identify which groups of points form single instances, i.e. individual cars. In Figure 3.4, the vehicle point cloud is shown before (on the left) and after (on the right) this instance clustering step. Each point is attributed an instance number and coloured based on this ID.

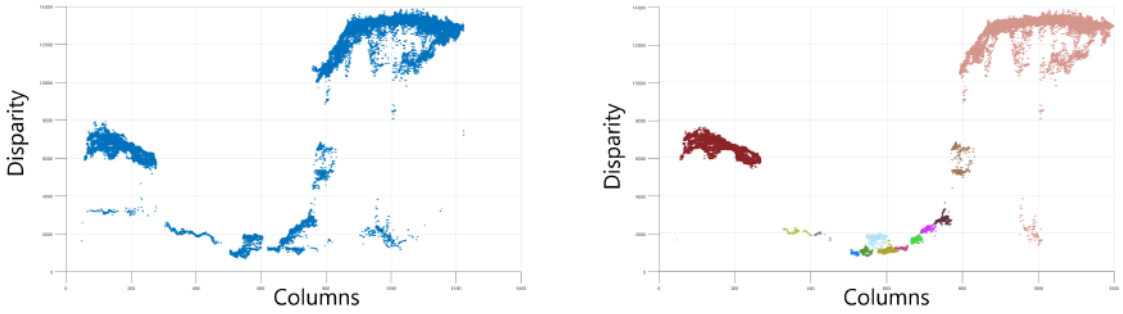


Figure 3.4: Vehicle point cloud before (left) and after(right) instance clustering.

Unlike the segmentation in step 1, cluster analysis falls under the category of unsupervised machine learning and discover natural pattern and groups in the given data. By performing clustering on a point cloud of a single class, the feature **class**

is ignored to simplify the clustering problem, leaving only the features that relate to space, i.e. the geometry of the scene.

Explicitly, the geometry features are the combination of the position in the image (row, column) and the disparity value. The data used as a feature for a clustering algorithm is crucial: it can influence the algorithm either positively by grouping the points in a coherent spatial fashion or negatively by not providing the right spatial discriminating criteria. In fact, there is an inverse proportional link between disparity and distance, so using the 3D coordinates relative to the ego-vehicle, X, Y and Z as features is explored in Section 5.4.1. Different types of clustering are also investigated in Chapter 5.

3.2.5 Step 5 - Mapping

Once the points are clustered, camera projective geometry is used to reconstruct the 3D scene in the ego-vehicle coordinate frame from the image pixels. Each instance of the vehicle class is therefore re-projected and directly represented by its silhouette on the 3D map. This is more representative of the actual space occupied by the vehicle in reality, compared to a standardised and generic 3D bounding box used in the state of the art approaches.

It is important to note that despite the mathematical formalisation of the 3D projection made in Section 2.3.1, the camera projection is dependent on the hardware set-up used and its quality relies heavily on the camera calibration process. The equations described in this section are therefore taken from the work in the Cityscapes dataset [65], which is used in the rest of this thesis. The camera coordinate frame is defined following ISO8855, where the X and Y axes are parallel to the ground observing the right-hand rule with the X axis pointing forward of the ego-vehicle in the driving direction. The Z-axis is therefore pointing up.

The calibration files of the dataset contain the intrinsic camera parameters: the focal length in pixel (f_x, f_y) and the optical centre (u_0, v_0). The resulting intrinsic matrix in

coordinate frame of the camera is:

$$C = K \cdot \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} u_0 & -f_x & 0 \\ v_0 & 0 & -f_y \\ 1 & 0 & 0 \end{bmatrix}. \tag{3.1}$$

The extrinsic parameters are the pose of the camera in the vehicle coordinate frame, consisting of the translation vector t and the rotation matrix R. The position is given by $t = [x, y, z]^T$ and the orientation is given using yaw ψ, pitch θ and roll ϕ, in radians. The rotation matrix R from the camera coordinate frame to the vehicle frame is computed with a rotation around roll axis, pitch axis and then the yaw axis. It is expressed as follows:

$$R = \begin{bmatrix} cos\psi cos\theta & cos\psi sin\theta sin\phi - sin\psi cos\phi & cos\psi sin\theta cos\phi + sin\psi sin\phi \\ sin\psi cos\theta & sin\psi sin\theta sin\phi + cos\psi cos\phi & sin\psi sin\theta cos\phi - cos\psi sin\phi \\ -sin\theta & cos\theta sin\phi & cos\theta cos\phi \end{bmatrix} \tag{3.2}$$

The distance estimated with stereovision is given in the camera coordinate frame and noted x_c. Using the intrinsic camera parameters, the baseline b between the two cameras and the disparity d, a pixel (u, v) in the image is projected to the point $P_c = [x_c, y_c, z_c]^T$ in the camera frame with the following equation:

$$\begin{aligned} x_c &= \frac{f_x \times b}{d} \\ y_c &= \frac{(u_0 - u) \times x_c}{f_x} \\ z_c &= \frac{(v_0 - v) \times x_c}{f_y} \end{aligned}$$

The resulting coordinates in the ego-vehicle frame is then given using the extrinsic camera parameters:

$$P_v = \begin{bmatrix} R & t \end{bmatrix} \cdot \begin{bmatrix} P_c \\ 1 \end{bmatrix} \tag{3.3}$$

An example of the resulting point cloud in the ego-vehicle coordinate frame is shown in Figure 3.2 under step 5 and in Figure 3.3. Each cluster is attributed a random colour for visualisation.

3.3 Experiment

A preliminary experiment was conducted in order to assess the behaviour of the pipeline on real data. In this section, I present the methods used and the selection of a dataset for traffic scenes understanding. Available evaluation metrics are also described and discussed. Because of the innovative way this proposed work represents object targets, appropriate evaluation measures do not exist yet due to a lack of ground truth data.

Finally, I present how I created a target data based on the ground truth provided by the selected dataset. This target data illustrates the ideal 3D panoptic segmentation that should be achieved and will serve as reference to compare the output of the framework.

3.3.1 Methods

In this section, I outline the methods selected for the different steps of the framework: semantic segmentation, disparity estimation and clustering.

Step 1. For semantic segmentation, the deep neural network SegNet [41] is used. The network is trained on the Cityscapes dataset [65] to classify 11 classes. SegNet's architecture benefits from saving the indices of the max pooling layers, making its decoder part more accurate in locating features. The CNN used in the encoder part is the VGG16 [40]. More information can be found in Chapter 4.

Step 2. The disparity maps are pre-computed and provided by the Cityscapes dataset. The method applied is Semi-Global Matching (SGM) [69] based on the implementation of [70].

Step 3. The filtering step aims to clean the point cloud that is created with the *vehicle* class. As the accuracy in depth estimation using stereovision is limited with distance, the problem is restricted here to the first 50 m. Points beyond this distance are filtered out.

Step 4. The clustering step is performed with the data formatted as *row, column, disparity*. Gaussian Mixture Model (GMM) is employed using the iterative Expectation-Maximisation (EM) algorithm. GMM is a technique based on the data distribution and requires as input that a co-variance matrix be specified. Considering the dynamic nature of urban traffic, the expectation is that vehicles appear in different sizes and shapes in the data. Consequently, it is logical to assume that clusters' co-variances should not be shared. Co-variances matrices are set to be diagonal.

To summarise, the composition of the pipeline for this preliminary test is the following:

1. Semantic Segmentation: SegNet
2. Disparity map: Semi-Global Matching
3. Filtering: keeping *vehicle* points within 50 m
4. Clustering: Gaussian Mixture Models
5. Mapping: based on the Cityscapes dataset

This test is presented in Section 3.4.2 as a case study computed on a subset of the dataset selected in the following section.

3.3.2 Available datasets

Data is more than ever at the centre of algorithm development due to the increasing use of supervised machine learning techniques, where AI learns from provided data. In my previous work [58], the algorithm was tested on data that I collected myself. However, despite existing tools to facilitate images labelling [71], the collection of data and its labelling can be time consuming and effort demanding.

Over the years, efforts have been made to create very large datasets for image recognition and object detection [67] [72] [73] [45]. As the performance of the algorithms improves and the nature of the tasks evolves, the datasets are becoming more specific to the applications for which they are created.

Autonomous driving is no exception, and I provide in this section a non exhaustive list of datasets of images available for traffic scenes understanding. Some are provided with benchmarks for different type of tasks with their corresponding metrics, which

will be discussed in the next section. Examples of tasks related to autonomous driving include depth estimation, optical flow, object detection and tracking, segmentation, and more.

Note that there is an effort from the research community to provide some datasets which are compatible with each other even though the nature of the data provided can differ. The motive is to build up on previous work and expand the pool of examples for the machines to learn. Also note that some benchmarks are flexible enough to evolve over the years as the domain progresses by adding tasks and appropriate labels for their data.

At the time of starting this research, the most appropriate and known datasets in visual scene understanding for autonomous driving are the following:

- Cityscapes [65]
- CamVid [74]
- KITTI [75] [64]
- Daimler Urban Segmentation [76]
- MATLAB Automated Driving Toolbox [77]
- SYNTHIA [78]
- ParallelEye [79]

For the purpose of evaluating the framework developed in this work, labels are necessary for semantic segmentation, instance segmentation and available depth information. Among them, the first two were especially produced for semantic segmentation and therefore provide pixel-wise annotated images. While KITTI did not have this labelled data originally for semantic segmentation, it was later added [80]. KITTI has the advantage to supply LiDAR measurements, whereas CamVid does not have depth information and the Cityscapes provides pre-computed disparity maps. Due to its advantageously large number of images, the Cityscapes was selected for this work.

More recently, supplementary material was released by these established datasets and several new datasets were released by autonomous vehicles companies to con-

tribute to the development effort. Notably, Berkeley DeepDrive published a very diverse dataset [81], the Cityscapes dataset released a 3D version of its data with panoptic segmentation annotations [82], and KITTI released a semantic labelling of its LiDAR measurements [83] [84]. Moreover, data from companies like Waymo [85], Lyft [86] and Audi [87] also contain information for other tasks than environment perception.

However, it is important to note that none of this later published data was available at the time of this research.

3.3.3 Evaluation

In this section, appropriate metrics are explored in order to evaluate the accuracy of the output of the developed framework for 3D panoptic segmentation. This section is not meant to describe every metric in detail. Performance measures are often linked to the label format of the data as well as the task performed. Benchmarks agree upon standard formats and metrics for comparability.

Generally, for classification tasks, existing metrics involve a confusion matrix, from which can be derived other related metrics to quantify different aspect of the algorithm such as global accuracy, precision, recall, F-score, and Receiver Operating Characteristic (ROC) curves.

For semantic segmentation, more appropriate metrics emerged like the Intersection over Union (IoU) and the Boundary F-score (BF) [5]. For object detection, the accepted criterion, established by the PASCAL VOC Challenge [72], is the average precision (AP).

As instance segmentation combines both segmentation and object detection, the metrics commonly used are both IoU and AP. Average precision is usually given with different variants (AP50, AP75, AP[0.5:0.95]), which vary depending on the acceptance threshold based on the IoU metric. Most metrics for object detection also consider the confidence score, which is the probability of the detection to be a correct detection and is usually produced by the neural network. In this work, traditional object detectors are not used and thus no confidence score is produced, which limits the usage of traditional metrics.

Besides, external validation metrics require a labelled version of the data in use. In the designed pipeline, 2D image analysis is combined to obtain a 3D point cloud and the results are in 3D space. On the contrary, all labelled data for semantic and instance segmentation in the datasets mentioned above is designed for 2D images.

Furthermore, 3D object detection is evaluated with 3D bounding boxes, which the developed framework does not produce either as it outputs directly the profile of each instance. In KITTI, 3D profile information comes from LiDAR data but is not used for evaluation. Indeed, the obtained point cloud in the framework also depends on the segmentation performance, and thus is not always directly comparable point by point.

As there is no corresponding and appropriate evaluation measures for the output of the created framework, this study will give a qualitative evaluation. For comparison purposes, the different 2D ground truths of the Cityscapes that were available at the time of the study are merged to produce the target data: the supposedly correct aspect of the 3D panoptic segmentation.

3.3.4 Target data

In this section, the generation of the target data for qualitative evaluation is presented. This target data is created to visualise the correct aspect of the data of the Cityscapes once projected in 3D in the vehicle coordinate frame. To this end, I combine the disparity maps with the ground truth provided by the Cityscapes dataset.

The ground truth is comprised of pixel-wise class labels, instance labels and polygons labels for each instance. However, the instance labels do not specify a different number for each instance; instead it highlights the pixels belonging to an instance and gives its class as pixel value.

In order to produce pixel-wise instance labels with an individual instance number instead, the polygons in JSON format are used and read with [88]. Each polygon file contains for the corresponding image the width and height of the image in pixels and its “objects” list. Objects are listed from back to front in the image with the object’s class label and the list of points forming the instance’s polygon in the format $[col, row]$.

All objects from the *vehicle* category are extracted: *car, truck, bus, caravan, trailer,*

train or motorcycle. As the polygon describes the exterior border of the objects, if there is an occluding object in the foreground, these pixels will be included in the target data and marked as belonging to the *vehicle* instance even if this occluding object is not from this category. To prevent this, the extracted polygons are then combined with the pixel-wise semantic segmentation ground truth to remove any points that are not from the *vehicle* category.

The target data at this stage is for 2D panoptic segmentation purposes: combining semantic and instance segmentation, with an emphasis on the *vehicle* class to fit the experiment. In order to have a target data that contains depth information and is projected in 3D, the generated 2D panoptic segmentation ground truth is merged with the disparity maps available in the Cityscapes dataset.

The resulting target data shows what an ideal output of the developed pipeline should look like. However, the disparity maps provided are not meant to be a ground truth and thus, they are not representing the actual distance of the vehicles in the scene. Consequently, this target data cannot be considered an ideal ground truth and will not be used for quantitative evaluation. However, it provides the best estimate of a 3D panoptic segmentation that can be achieved. This will be essential to support qualitative comparison and illustration purposes, and to assume a perfect semantic segmentation in Chapter 5.

Another consequence is that, due to imprecision in the disparity maps, the target data suffers from similar errors as in step 3 of the framework in Section 3.2.3. Consequently, an equivalent filtering is applied here, with the difference that the morphological operation is reduced as the disparity reduces and is limited by the size of the *vehicle* instances.

The framework was initially tested on a subset of the Cityscapes dataset: the city Lindau in particular, which includes 59 images. Finally, following the steps description in Section 3.3.1, the target data is filtered to withdraw points beyond 50 m. Examples of the target data are shown in Figure 3.5.

From left to right: left image, ground truth segmentation, disparity map, vehicle point cloud, filtered vehicle point cloud.

Figure 3.5: Target data generation

For visualisation purposes and to facilitate the comparison of the pipelines' outputs, all point clouds in this work are displayed the same way as the target data: in the ego-vehicle coordinate frame. To complement the presentation of the results, for each scenario presented, the original colour image of the left camera and the corresponding target data will always appear beside the evaluated point cloud. All the results would be better appreciated when visualised on a colour display.

For the point clouds in the ego-vehicle coordinate frame, 4th and 5th columns in Figure 3.5, the axes represent the different 3D directions: forward, the sides - left/right - and the height - up/down, denoted by $[X, Y, Z]$ respectively in the ISO denomination used by the Cityscapes dataset as mentioned in Section 3.2.5.

It is important to note here that the axis forward is defined relative to the camera, which means it is not always aligned with the road up ahead, in particular in curves. This is why the points observed do not systematically go along the forward axis, as seen in scenario presented on row 1 and 5. The ego-vehicle position is represented by the larger white dot at the position $[0, 0, 0]$.

It should also be noted that the colours of the clusters will not always match between the target data and the resulting output of the pipeline for couple of reasons. Firstly and most importantly, the colours are randomly generated. Secondly, the clusters in the target data are ordered back to front, but the clustering algorithm in the pipeline does not necessarily find them in that same order. Additionally, it will be shown and explained later in this work that some algorithms do not systematically find the same number of clusters.

3.4 Results and discussion

In the absence of an appropriate ground truth for quantitative measure on 3D data, I present as a case study how the pipeline behaves on a subset of the Cityscapes dataset. How this behaviour evolves dependent on the parameter and different method used, is explored throughout the rest of this document. Before presenting the preliminary results with the selected methods in Section 3.3.1, some observations are made about the generated target data.

3.4.1 About the data

Although rare, some errors occurred during the target data generation, which are detailed in this section.

Firstly, the car is counted twice because the polygon is actually defined twice in the file. This incorrectly increases the car counter and will be problematic in Section 5.1, where the clustering method needs to be given the exact number of vehicles to be found in the image. The data itself is not incorrect though, so the point cloud is not altered.

Secondly, polygons of small instances are sometimes non-existent. Possibly, they are too small to be uniquely identified by the annotators. These instances are either labelled “car group” or have been merged with the closest car when it is an overlapping instance. The dataset could be altered and corrections shared with the original authors for review, however, this issue is inevitable as the spatial resolution of a camera is finite.

Finally, removing non-vehicle pixels from foreground objects causes issues with transparent or go-through objects; a fence for example. Indeed, pixel labels cover the entire area of the fence, not just its bars. Unfortunately, this also removes car pixels that may be behind and sometimes whole instances.

As one can partially see through a fence, these vehicles could be detected and recognised by the neural network. However, the IoU performance metrics will mark it wrong as this is not the expected pixel class label. But was it wrong? A similar issue can arise when there is a conflict in the definition of the classes. Notably, bicyclists are defined inconsistently in different datasets.

This type of errors raises an interesting point of discussion for all the edge cases of this type found in the real world, in particular for the semantic segmentation task. Contrary to common beliefs, a neural network does not develop critical thinking, reasoning ability nor common sense but rather it consists of a complex function optimised to recognise statistical cues from what it is shown. It learns through experience but does not extract concept definitions. Despite being given billions of examples of what a vehicle pixel looks like, it does not know what a vehicle is per se.

Thus, if a neural network is told that the pixel is a fence while it is not, the neural network does not have the ability to know that this is a mistake. Instead, the neural network will believe that this is a fence pixel and adjust its experience. Although in practice, it is commonly accepted that the number of occurrence for errors of this type is too small to have an impact.

There is a need for appropriate standard in the artificial intelligence domain to handle dataset creation in terms of quality, exactness, unconscious biases, ethics and others. As of now, these standards are investigated and in development.

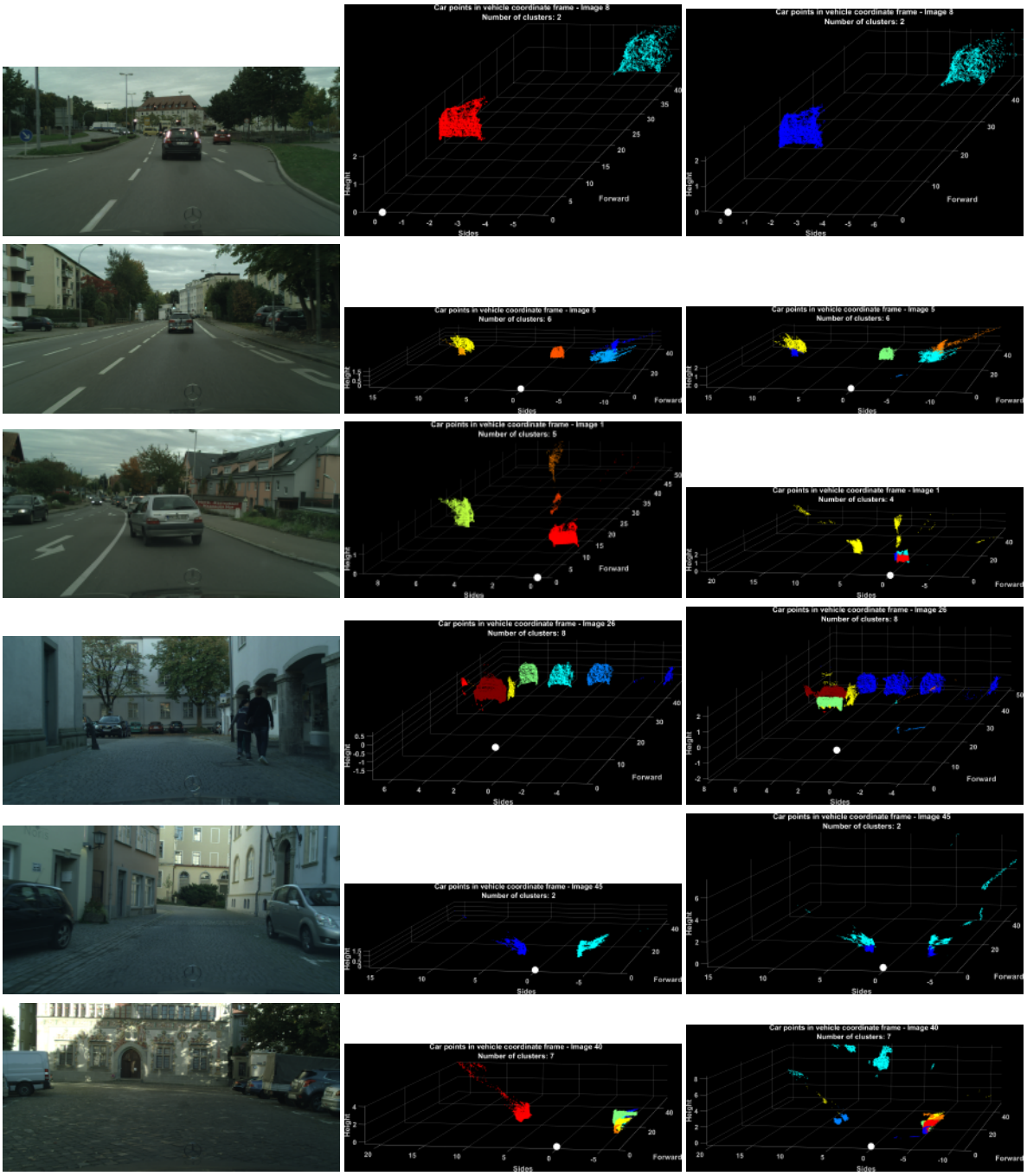
3.4.2 Preliminary tests

In this section, the results of the preliminary test are presented and demonstrate the viability of this pipeline. It will also help in identifying the potential benefits as well as the current limitations and improvement areas that will be discussed in the next Chapters. There are mainly two characteristics to observe: the aspect of the point cloud and the division into instances. The Figure 3.6 introduces scenarios that condense into a few examples the recurrent behaviours and patterns observed during result analysis.

In row 1 from Figure 3.6, the obtained point cloud is the same as the target data so the vehicle points are correctly detected and there are no outliers. Both vehicles are also properly identified, with the profile of the one immediately in front and in the same lane correctly mapped at 20 m. Row 1 illustrates that the pipeline can work on an ideal scenario: two-ways traffic with few and clearly separated vehicles in the middle of the field of view in front of the ego-vehicle.

In row 2, the result shows the pipeline has potential on a more complex scenario: more vehicles, most of them parked on the side of a two-way street and thus occluded in the image. As the vehicles are on the extremities of the image, the point cloud exhibits scattering on the sides. This can also be observed on the target data, they are incorrect value disparities. The cause can either be the accuracy of the disparity maps itself or the precision of the superposition with the semantic segmentation in step 3.

Except a group of points in the foreground, the point cloud resembles the target



From left to right: left image, target data, pipeline's output with SegNet16 and GMM

Figure 3.6: Preliminary results of pipeline

data and five out of the six vehicles are correctly detected, identified and located. This includes the vehicles that are considered high-priority: the vehicle in front in the same lane and the first parked vehicle on the side of the ego-vehicle's driving lane.

However, the other scenarios in Figure 3.6 show discrepancies so these promising results are not repeated and reliable. Resulting point clouds in row 3, 4 and 5 display added small groups of points, like noise, compared to the target data. This is likely to be a repercussion from the semantic segmentation. Indeed, points wrongly classified as *vehicle* are included in the point cloud and are too significant in size to be filtered out with the morphological operation from step 3.

This effect is particularly evident in row 6, where blobs of a relatively large size appear in the point cloud while it should not, as observed in the corresponding target data. Note that the scattering observed on the target data on row 6 can be similarly interpreted as the scattering in row 2.

In row 5, despite the two vehicles being clearly separated on the left and on the right, the clustering algorithm surprisingly divides them between top and bottom. I attempt to clarify this behaviour in Section 5.1.

Finally, in row 3 and 4, the separation into instances manifests the same type of response: a single vehicle in the foreground is unnecessarily divided into several instances whereas all the vehicles in the background are grouped as one cluster.

A suspected cause is a misunderstanding of the data distribution. Either the specified co-variance matrix for GMM is not adequate, or the clustering is affected by the variations in the features. Indeed, the variation of disparity values decreases as distance increases because distance is inversely proportional to disparity. This variation reduction can be seen in Figure 3.4. This is further investigated in Chapter 5.

To summarise, the pipeline I developed demonstrates promising results but tends to be irregular in performance. The version of the pipeline tested in this chapter is basic as no parameter tuning has been explored yet. It also uses a clustering method that will be shown later to not be the most suitable to this specific application. Thus, this preliminary test not only demonstrates the functionality of the proposed pipeline, but it allows the identification of limitations and especially improvements areas that will be studied in the coming Chapters.

Errors in the resulting point cloud come in mainly three types:

- issues with the cluster colour (ID) reveal a problem with step 4 of the pipeline:

clustering,

- issues with additional or missed points reveal a wrong detection and thus a problem with step 1 of the pipeline: semantic segmentation,
- issues with the points location reveal a problem with step 2 of the pipeline: disparity estimation.

The latter is not reviewed in this work but the mapping precision could be researched in further work; notably by computing the disparity maps, or directly the depth maps, with other techniques than SGM.

In Chapter 4, improving the point cloud aspect is addressed by studying the semantic segmentation task and determining predominant criteria for performance. In Chapter 5, the designation of instance's ID is investigated by demonstrating the limits of GMM for this application and changing the clustering method.

More generally, the innovative framework I developed differs greatly from current techniques and has not been tested before. The pipeline steps away from the traditional problem definition of a 2D instance segmentation on images by incorporating depth information to pursue the processing directly in 3D space.

It leverages the usage of both disparity maps and semantic segmentation to create a classified 3D point cloud of objects of interest in the scene. The proposed framework meets the characteristics listed in Section 3.1 by processing visual data with a range of techniques not fully relying on deep convolutional neural networks but also utilising clustering techniques on raw data to identify each of these objects.

The output of the pipeline provides a point cloud representation that fits closely the vehicles shape. This is by nature more precise than bounding boxes as the latter only highlight approximate areas that are larger than the actual vehicle size, and serve as "space holder".

Besides, the resulting point cloud of this framework is sparse, with hundreds of thousands of points on average. This does not hinder the ability of the framework to give a coherent and clear representation of the vehicles at a further distance. This is a very interesting feature as, for comparison, a LiDAR point cloud would consist of millions of points. Although, computation time is not evaluated in this work, it is worth

noting that this could be an advantage.

Furthermore, there is still room for optimising the pipeline in steps 1 and 2, semantic segmentation and disparity map respectively, as they can be computed independently from each other. Therefore, they can be computed in parallel for speed. Similarly, the clustering step could be performed for other dynamic objects such as pedestrians and bicyclists separately.

3.5 Summary

In this chapter, the objective of this thesis was presented: developing a framework for 3D panoptic segmentation based on visual data only and leveraging unsupervised clustering methods as well as disparity information in a different manner than in the literature. The pipeline goes beyond approaches in the literature with well defined object profile located in the 3D space.

The proposed framework is described as a 5 step pipeline that performs instance clustering on a classified point cloud created from 2D image analysis. The instances' shapes are then mapped with respect to the ego-vehicle.

This innovative framework is tested with state of the art approaches on a dataset of urban traffic scenes appropriate for autonomous driving application. The methods of evaluation are explained and the target data generated for qualitative comparison is introduced.

The preliminary results manifest potential as the pipeline presents some advantages and successfully classifies, detects and identifies vehicle instances in some of the explored scenarios.

However, it also suffered some drawbacks, notably in the vehicle point cloud formation and in the instance clustering. Analysis of these limitations and possibilities for improvement are explored in the next chapters.

4. SEMANTIC SEGMENTATION

In the previous chapter, I presented the initial version of the developed framework that can perform vehicle instance segmentation on a 3D point cloud from a pair of colour images. This method differs greatly from current techniques in its way of combining existing data and because it steps away from the traditional problem definition of a 2D instance segmentation on images.

However, the approach suffers from several drawbacks, including that clustering techniques are sensitive to the features given as input. In this framework, these input features consist in the raw data of the vehicle point cloud. Therefore, the aspect itself of the point cloud matters significantly in the accuracy of the framework and the correct detection of vehicle points must be ensured.

As the vehicle detection is a result of step 1 of the framework, I explore in this chapter the influence of the pixel-wise semantic segmentation on the aspect of the vehicle point cloud. To do so, I investigate the depth of a particular network architecture, SegNet [41], and the complexity of the classification task. The results shows the benefit of using deeper networks with a simpler binary classification task rather than multiple objects classification. Findings also highlight the importance for my framework of a specific evaluation metric, the boundary F-score [5], which is a novelty in autonomous driving applications.

The chapter is divided as follows: I first describe the network model, its training and optimisation, the datasets used and the evaluation metrics. I then present the results on multiple objects classification, followed by the results on binary classification for a vehicle detector. Finally, I illustrate the effects it has on the vehicle point cloud.

4.1 Network architecture

The semantic segmentation task classifies and labels images at the pixel level. This means that for an RGB image given in input, the network outputs another image with a class label for each pixel. It should not be confused with instance segmentation, as semantic segmentation gives the same value to all pixels belonging to the same class, while instance segmentation gives a unique label to each object.

Among several architectures created for the purpose of semantic segmentation [89] [90] [91], I selected SegNet because it was primarily developed for autonomous driving applications using an efficient convolutional encoder-decoder network. Its architecture is shown in Figure 4.1.

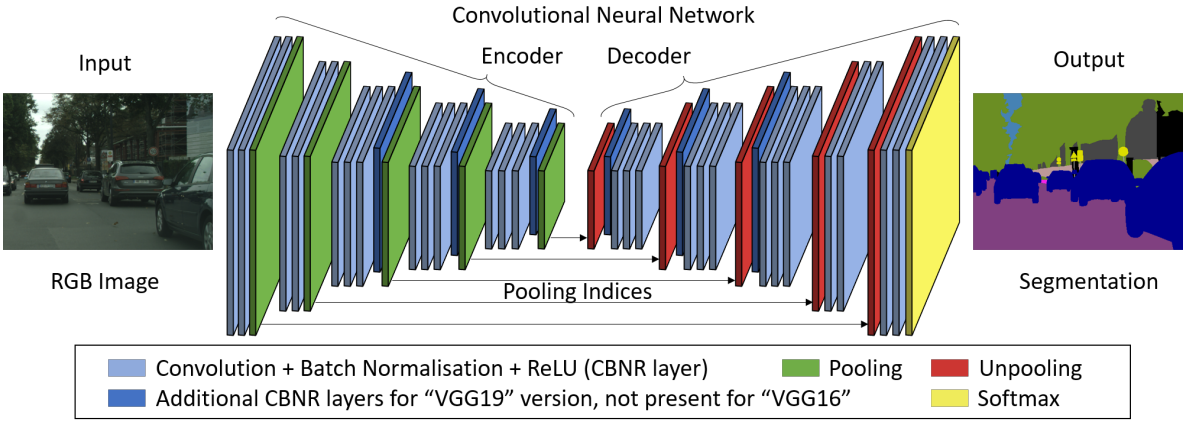


Figure 4.1: Architecture of SegNet

An encoder-decoder network is a symmetrical network. The encoder layers correspond to feature extractors, similar to the ones found in convolutional networks used for image recognition, as explained in Chapter 2. In SegNet, the encoder architecture adopts the thirteen convolutional and five pooling layers of the VGG16 [40]. Each "convolutional" layer is actually comprised of:

1. convolutional layer, applying a convolution over the input image or the intermediate feature maps. The convolution corresponds to a filter [33], extracting features stored in the next layer.
2. batch normalisation layer [92], normalising the inputs in order to facilitate training and stabilise the learning process by reducing the internal covariate shift (change in the data distribution).
3. ReLU layer, which is the activation layer selecting only the features of interest to pass onto the next layer.

The decoder layers, on the other hand, upsample the feature maps so that the final dimensions of the network's output match the input image size.

The particularity of SegNet resides in the use of pooling indices. Each pooling layer

of the encoder reduces the feature maps' dimensions by a factor of two by using a 2 x 2 sliding window and keeping only the maximum value to the next layer. The location of this maximum value is stored by SegNet until the corresponding upsampling layer in the decoder phase. This upsampling layer creates a sparse features map with the feature placed at the right location and the deconvolution layers are trained to fill in the gaps to reach a dense features map.

It is this specificity that makes SegNet particularly suitable for real-time applications such as autonomous driving.

For my experiments, I use four variants of this architecture by changing two aspects:

- the depth of the network.
- the classification (softmax) layer.

4.1.1 Depth of network

VGG is an object classification network that comes in two versions: VGG16 and VGG19 [40]. The characteristic difference between the two is the number of convolutional layers: thirteen and sixteen respectively.

As illustrated in Figure 4.1, changing the encoder of SegNet from VGG16 to VGG19 is equivalent to adding six more layers due to the decoder mirroring the encoder architecture. The added layers, as well as the number of added learnable parameters, are listed in Table 4.1.

On one hand, adding more layers to a network brings more parameters to the model. Deeper networks allow to capture and extract more complex features in images, which has shown to improve their classification. On the other hand, the addition of parameters also poses a greater risk of overfitting the training data. This could hinder the ability of the network to generalise on unseen images.

In my experiments, I compare between SegNet with the VGG16 encoder and SegNet with the VGG19 encoder in order to determine if a deeper architecture also improves the pixel-wise semantic segmentation of images.

Total Learnables for SegNet with VGG16 encoder						**29,449,377**
		Layer	Type	Learnables		Total Learnables
Added layers	Encoder	3_4	Convolution	Weights	3x3x256x256	590,080
				Bias	1x1x256	
			Batch Normalization	Offset	1x1x256	512
				Scale	1x1x256	
			ReLU	-		0
		4_4	Convolution	Weights	3x3x512x512	2,359,808
				Bias	1x1x512	
			Batch Normalization	Offset	1x1x512	1,024
				Scale	1x1x512	
			ReLU	-		0
		5_4	Convolution	Weights	3x3x512x512	2,359,808
				Bias	1x1x512	
			Batch Normalization	Offset	1x1x512	1,024
				Scale	1x1x512	
			ReLU	-		0
	Decoder	5_4	Convolution	Weights	3x3x512x512	2,359,808
				Bias	1x1x512	
			Batch Normalization	Offset	1x1x512	1,024
				Scale	1x1x512	
			ReLU	-		0
		4_4	Convolution	Weights	3x3x512x512	2,359,808
				Bias	1x1x512	
			Batch Normalization	Offset	1x1x512	1,024
				Scale	1x1x512	
			ReLU	-		0
		3_4	Convolution	Weights	3x3x256x256	590,080
				Bias	1x1x256	
			Batch Normalization	Offset	1x1x256	512
				Scale	1x1x256	
			ReLU	-		0
Total Learnables for SegNet with VGG19 encoder						**40,073,889**

Table 4.1: Architecture difference in layers and parameters numbers between SegNet VGG16 and VGG19

4.1.2 Classification layer

The classification layer determines the output of the network and can be changed to accommodate for the classes present in the given classification task. Originally, SegNet was trained on the CamVid dataset [74], which contains 11 classes. However, multiple objects classification is a more complex task and more computationally heavy than binary classification. I therefore explore the original set-up with 11 classes - *sky, building, pole, road, sidewalk, vegetation, signs, fence, vehicle, pedestrian, bicyclist* - but also a specific detector for the *vehicle* class with the rest of the image content marked as *background*.

In MATLAB, the classification layer outputs the categorical label for each image pixel based on the probabilities of this pixel to belong to each class. These probabilities, represented by $\hat{y}_c$, are computed in the preceding layer: the softmax layer, in yellow in Figure 4.1, following Equation (4.1), where z_c is the output of the last deconvolution layer for class c.

$$\hat{y}_c = \frac{exp(z_c)}{\sum_{i=1}^{C} exp(z_i)} \tag{4.1}$$

Thus, the size of the last layers of SegNet is [ImageSize, 11] for multiple objects classification and [ImageSize, 2] for binary classification.

Once the architecture of the model is defined, the network is trained on relevant data so that the parameters of the network - the weights and biases - take optimum values for the task it is going to be used for. As most of the deep learning libraries or toolboxes now, the backpropagation algorithm [93] necessary to train a neural network is already handled in them. Thus, the classification layer here also defines some of the training options, in particular the loss function and the class weights.

The loss function measures how well the network is performing on each example in the data. After a forward pass, the network prediction $\hat{y}$ is evaluated against the corresponding ground truth y using Equation (4.2) for binary classification and Equation (4.3) for multiple objects classification. This is the cross-entropy loss, which effectively maximises the probability of the correct class.

$$\mathscr{L}(\hat{y},y) = -[ylog(\hat{y}) + (1-y)log(1-\hat{y})] \tag{4.2}$$

$$\mathscr{L}(\hat{y},y) = -\sum_{c=1}^{C} y_c log(\hat{y}_c) \tag{4.3}$$

Ideally, the network should see during training a wide variety of examples with a fair representation of each class so that each have a balanced number of observations. Indeed, when the training data is imbalanced towards a specific class, the network develops a bias towards this dominant class and tends to predict that class by default.

For pixel-wise semantic segmentation applied to autonomous driving, traffic scenes often show this imbalance due to the area covered by each class in images: there are more pixels of sky, road or buildings than of pedestrians for example.

To prevent important biases in the learning process, I apply class balancing by adding weights to the classification layer. The weights for each class are linked to their appearance frequency in order to mitigate this imbalance in the data, as described in [94].

4.2 Training

As mentioned in the previous section, the objective of the training is to find the optimum values for the parameters of the network - weights w and biases b. The predictions $\hat{y}$ of the network can be expressed mathematically with these parameters w and b. Combining the loss $\mathscr{L}$ for each example in a dataset of m examples, the error made by the network on the training data is calculated by the cost function in Equation (4.4).

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathscr{L}(\hat{y}^i, y^i) + \frac{\lambda}{2}||w||_2{}^2 \tag{4.4}$$

Reducing the error made by the network becomes an optimisation problem: finding the parameters w and b that minimise the cost function J.

The second term in Equation (4.4) is a regularisation term added to the cost function

in order to prevent overfitting. Intuitively, by minimising the L2 norm of the weights, some neuron units would not pass the activation function anymore. This simplifies the network. The regularisation factor λ is set to 0.0005 in my experiments.

Note that the training process described in the following sections, unless stated otherwise, is the same for all four declinations of SegNet that I compare.

4.2.1 Weights optimisation and initialisation

To minimise J, I use the gradient descent in which the weights are updated after each training iteration by taking steps towards the negative gradient of J. The size of the step is decided by the hyperparameter α, the learning rate, initialised to 0.001 in my experiments.

In order to make the convergence towards the optimum faster and reduce the oscillations on the way, I use the gradient descent with momentum. With momentum, the previous gradients are taken into account in order to find the steepest path to the minimum. The contribution of the previous step is defined by γ, which I set to the most commonly used value in the domain: 0.9.

The weights are updated as follow, where i is the current iteration:

$$w_{i+1} = w_i - \alpha \nabla J(w_i) + \gamma(w_i - w_{i-1}) \tag{4.5}$$

I also use mini-batch gradient descent rather than stochastic gradient descent as it updates the weights more often, making the convergence faster and generally closer to the minimum too. I set the mini-batch size to 2 because of GPU memory constraints. The weights are thus updated after seeing two training examples instead of after seeing the entire training set.

To initialise the parameters of the model, I use pre-trained weights. This is called transfer learning. Taking a model already trained for classification on a larger and more generic dataset helps in learning generic filters for images that can be adapted to another task later. Fine-tuning the parameters to a specific applications or another dataset requires less training and less data than training again from scratch. This is useful when training is constrained by either data, time, or equipment.

The encoder weights were therefore pre-trained on ImageNet [45], a very large collection of images (14 millions) for object recognition. SegNet's authors also trained the model further on CamVid dataset for the semantic segmentation of urban scenes. I fine-tune the training on semantic segmentation to accommodate for a larger dataset of urban scenes and for the changes that I made on the architecture, presented in Section 4.1.

4.2.2 Data

The models were trained on the Cityscapes dataset [65], which is a large dataset of urban traffic scenes: more than 5,000 finely annotated images and 20,000 coarse labelled images. I divided the finely annotated images into a training set of 3,640 images (52%), a validation set of 1,000 images (14%) and a test set of 2,310 images (34%).

The subsets were determined by the cities comprised in the datasets, respecting the balanced distribution established by the authors, and using the commonly accepted ratios for these subsets as a guideline. In order to provide more examples for the training, I used data augmentation by translating the images in both directions and flipping them horizontally.

The network described in Section 4.1 takes as input an image of size 360 by 480. As the images of the Cityscapes have a higher resolution (1024 by 2048), the images are cropped so that it fits the network while still benefiting from the image quality.

When trained for multiple objects classification, the networks were trained over the entire training set 50 times (50 epochs). For binary classification, the models were trained for 25 epochs.

Several hyperparameters were tried and I retained the values and model that led to the best result on the validation set. The values are mentioned in the sections above when appropriate. The validation set is also used to check the absence of overfitting after training convergence.

Note that during training with validation, memory errors occurred even on an HPC (High Performance Computing), halting the process. I reduced the number of validation examples to 250 in order to prevent another memory crash. After investigation

in collaboration with MATLAB, I found out that the implementation did not reflect the chosen parameter for the validation mini-batch size.

In spite of the parameter being set to the minimum possible, 2, the total number of validation examples would be passed on instead of the number of images in the mini-batch. Unfortunately, this issue forfeited the use of mini-batches and limited every user by their hardware set-up. The error was therefore fixed immediately and the correction effective in the following MATLAB release in 2019.

Both encoder variants of SegNet were tested for multiple objects classification and then for binary classification on the Cityscapes dataset, but also on KITTI [75], and CamVid [74]. These datasets contain fewer examples than the Cityscapes dataset, and I used respectively 132 and 233 test examples from these datasets. Authors of KITTI and Cityscapes collaborated in order to have matching classes' definitions and therefore compatible datasets. Testing on KITTI while having trained on the Cityscapes is another way to test the ability of the network to generalise what it has learned to unseen scenarios.

All trainings for multiple objects classification were done using an HPC with 4 GPU NVIDIA TESLA K80 and 128GB of shared RAM. All trainings for binary classification were done with a GPU NVIDIA Quadro K2200 4GB and 16GB RAM.

SegNet's implementation later accepted higher resolution images as input, so segmentation predictions have been reproduced at the original size of the Cityscapes dataset (1024 by 2048) to be used in the pipeline. This was computed with a GPU NVIDIA GeForce RTX2080 8GB and 64GB of RAM.

4.3 Metrics

In this section, I briefly describe the evaluation metrics that are used in the rest of the chapter.

In classification task within supervised learning, the use of confusion matrices to have a quantitative return on the algorithm performance is standard. In a confusion matrix, for a given class, the following are defined:

- True Positives (TP), which are correct detection,

- True Negatives (TN), which are correct rejection,
- False Positives (FP), which are false alarms,
- False Negatives (FN), which are missed detections.

The semantic segmentation task can be evaluated with the standard global accuracy, which is the number of correctly classified pixels over the total number of pixels, regardless of any class. However, this metric is not meaningful in segmentation, particularly when there is imbalance like in autonomous driving datasets.

The mean accuracy is the ratios of correctly classified pixels in each class over the total number of pixels belonging to that class, as shown in Equation (4.6), and then averaged over all classes.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.6}$$

The mean intersection over union (IoU) is the Jaccard index averaged over all classes. It quantifies how similar the segmentation and the ground truth are. In other words, it is the ratio of correctly classified pixels over the number of pixels assigned that class in the ground truth and the prediction, as in Equation (4.7). The weighted IoU is weighted by the number of pixels in the class.

$$IoU = \frac{TP}{TP + FP + FN} \tag{4.7}$$

The BF score, for boundary F1 measure, evaluates how similar segmentation contours are to the ones of the ground truth. The harmonic mean (F1 measure) is computed using precision and recall within a distance error tolerance of the ground truth boundaries.

$$F_1 = 2\frac{Precision * Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN} \tag{4.8}$$

According to Csurka *et al.* [5], the IoU along with the BF score quantify a segmentation result the most appropriately.

4.4 Multiple object classification

I present in this section the results of the pixel-wise semantic segmentation on a multiple classes problem. Recall the chosen 11 classes for the autonomous driving application are the following: *sky, building, pole, road, sidewalk, vegetation, signs, fence, vehicle, pedestrian, bicyclist*.

I compare on this task two encoder variants of SegNet: one with VGG16 and the other one with VGG19. Specifically, this is analysing the influence of the depth of the encoder-decoder on the task, with 6 more layers and 10.6 millions added parameters (an increase of 36%).

The results are first shown for the entire test sets, then by class, and finally, image by image. As the depth of the network does not show a particular effect on the metrics and therefore the quality of the segmentation, I analyse the training and validation.

4.4.1 Results and discussion

The evaluation of 11 classes semantic segmentation on the test sets are shown in Table 4.2. Globally, the addition of the 6 layers reveals a small improvement (+ 1.3 percentage point (pp) on average) on the Cityscapes dataset and on KITTI for global accuracy and mean BF score. This means that overall, the segmentation contours are more precise and the total number of correctly classified pixels is slightly better on these datasets.

However, the mean IoU does not show the same level of improvement, which tends to point out that the segmentation is not necessarily and significantly improved. In fact, the segmentation stays noisy.

In addition, the test on CamVid clearly demonstrates a deterioration of the segmentation when adding more layers, regardless of the metric used.

With more parameters, the neural network can capture more complicated models and functions, helping with the generation of appropriate features detectors. As the network was trained on Cityscapes only, these filters adapted themselves to recognise features in the images of this dataset in particular. The Cityscapes dataset and KITTI are two compatible datasets in terms of classes' definitions and both originate from European roads, German ones specifically.

	VGG16 → VGG19			Average improvement
	Tested on: (number of images in dataset)			
	KITTI (132)	CamVid (233)	Cityscapes (2,310)	
Global Accuracy	81.47% → 82.93%	79.99% → 71.60%	87.46% → 88.68%	-1.90
	+1.46	-8.39	+1.22	
Mean IoU	46.77% → 45.84%	46.43% → 38.91%	62.61% → 62.85%	-2.74
	-0.93	-7.52	+0.24	
Mean BF score	51.36% → 53.19%	51.09% → 44.89%	64.04% → 64.89%	-1.17
	+1.83	-6.20	+0.85	

Table 4.2: Comparison between SegNet with encoder VGG16 and SegNet with encoder VGG19. Datasets metrics for 11 classes.

This can explain why adding layers, and hence parameters, manifested a positive enhancement on these two dataset only. CamVid, on the other hand, is composed of British streets, with classes defined differently. Most notably, while CamVid labels *pedestrian* and *bicyclist*, Cityscapes defines *person*, *rider* and *bicycle* in order to remove the ambiguity. This difference obviously bring conflicts during training and evaluation.

The above results indicate that adding more layers increases the specificity of the neural network and that generalisation to unseen or to a wider range of scenarios becomes more challenging. Averaged on all datasets, the metrics decrease as well, which confirms this interpretation.

Following the dataset metrics, results by classes for the IoU and the BF score are in Table 4.3 and Table 4.4 respectively. Both tables demonstrate that the addition of layers does not systematically yield a more accurate segmentation.

Consistent with the observations made above, the IoU and BF score for most of the classes tested on CamVid are impaired with the additional layers. The only exception for both metrics is the *sign* category, which can be explained by the fact that the vast majority of signs in Europe, including the UK, follow European design norms so they are fairly generic and unvaried. Classes in which the change of environment, from Europe to UK, can be reflected have been particularly affected: *buildings, vegetation, sidewalks*.

Considering the training was done on Cityscapes, I examine the results on Cityscapes and KITTI more closely. Classes that showed consistent worsening with the

		Intersection over Union: VGG16 ⟶ VGG19		
		Tested on: (number of images in dataset)		
		KITTI (132)	CamVid (233)	Cityscapes (2,310)
1	Sky	83.710% ⟶ 84.507%	82.768% ⟶ 81.873%	87.724% ⟶ 86.800%
		+0.797	-0.895	-0.924
2	Building	59.150% ⟶ 55.061%	67.820% ⟶ 49.862%	80.546% ⟶ 78.229%
		-4.089	-17.958	-2.317
3	Pole	37.741% ⟶ 31.969%	18.160% ⟶ 9.158%	36.620% ⟶ 30.526%
		-5.772	-9.002	**-6.094**
4	Road	76.339% ⟶ 82.981%	75.952% ⟶ 72.376%	80.334% ⟶ 88.927%
		+6.642	-3.576	+8.593
5	Pavement	22.021% ⟶ 22.835%	43.628% ⟶ 21.795%	48.568% ⟶ 60.333%
		+0.814	**-21.833**	**+11.765**
6	Vegetation	79.155% ⟶ 77.394%	61.133% ⟶ 52.071%	83.491% ⟶ 83.189%
		-1.761	-9.062	-0.302
7	Sign Symbol	37.383% ⟶ 35.739%	30.363% ⟶ 31.940%	42.659% ⟶ 40.458%
		-1.644	**+1.577**	-2.201
8	Fence	16.645% ⟶ 24.375%	32.308% ⟶ 27.134%	29.133% ⟶ 26.290%
		+7.730	-5.174	-2.843
9	Vehicle	66.941% ⟶ 69.938%	59.340% ⟶ 53.092%	84.500% ⟶ 85.396%
		+2.997	-6.248	+0.896
10	Pedestrian	29.678% ⟶ 16.034%	32.587% ⟶ 22.435%	64.740% ⟶ 63.036%
		-13.644	-10.152	-1.704
11	Bicyclist	5.691% ⟶ 3.400%	6.617% ⟶ 6.303%	50.361% ⟶ 48.209%
		-2.291	-0.314	-2.152
Global IoU		46.769% ⟶ 45.839%	46.425% ⟶ 38.913%	62.607% ⟶ 62.854%
		-0.930	-7.512	+0.247

Table 4.3: Comparison between SegNet with encoder VGG16 and SegNet with encoder VGG19. Class IoU.

		Boundary F-score: VGG16 ⟶ VGG19		
		Tested on: (number of images in dataset)		
		KITTI (132)	CamVid (233)	Cityscapes (2,310)
1	Sky	78.157% ⟶ 79.895%	75.825% ⟶ 72.837%	84.795% ⟶ 83.960%
		+1.738	-2.988	-0.835
2	Building	33.524% ⟶ 31.165%	57.869% ⟶ 43.938%	65.646% ⟶ 62.955%
		-2.359	-13.931	-2.691
3	Pole	55.034% ⟶ 50.443%	50.959% ⟶ 37.461%	59.345% ⟶ 52.881%
		-4.591	-13.498	**-6.464**
4	Road	54.844% ⟶ 65.525%	56.738% ⟶ 57.946%	67.973% ⟶ 77.782%
		+10.681	+1.208	**+9.809**
5	Pavement	31.819% ⟶ 38.366%	59.713% ⟶ 59.202%	52.271% ⟶ 61.835%
		+6.547	-0.511	+9.564
6	Vegetation	64.456% ⟶ 64.247%	53.434% ⟶ 36.227%	72.797% ⟶ 71.725%
		-0.209	**-17.207**	-1.072
7	Sign Symbol	36.713% ⟶ 37.009%	37.431% ⟶ 39.121%	51.189% ⟶ 49.703%
		+0.296	**+1.690**	-1.486
8	Fence	27.230% ⟶ 33.234%	29.626% ⟶ 27.037%	32.462% ⟶ 32.581%
		+6.004	-2.589	+0.119
9	Vehicle	51.871% ⟶ 55.288%	41.987% ⟶ 37.159%	75.269% ⟶ 76.359%
		+3.417	-4.828	+1.090
10	Pedestrian	43.144% ⟶ 30.679%	40.315% ⟶ 31.479%	61.056% ⟶ 60.718%
		-12.465	-8.836	-0.338
11	Bicyclist	25.937% ⟶ 22.214%	25.503% ⟶ 25.123%	48.432% ⟶ 48.221%
		-3.723	-0.380	-0.211
Global BF score		51.359% ⟶ 53.185%	51.094% ⟶ 44.892%	64.038% ⟶ 64.889%
		+1.826	-6.202	+0.851

Table 4.4: Comparison between SegNet with encoder VGG16 and SegNet with encoder VGG19. Boundary F-score.

addition of more layers on both IoU and BF score are: *building, pole, vegetation, pedestrian and bicyclist*.

They are either classes of thin objects or classes with varied textures. On one hand, the unimproved segmentation of thin objects can be explained by the fact that the deeper the network, the more spatial resolution is lost through the pooling layers as the feature maps dimensions reduce. However, the pooling indices saved by SegNet should have mitigated this effect.

On the other hand, adding layers in the encoder, which is the feature extractor, should have helped in capturing textures. Indeed, the first layers are equivalent to basic filters, while the deeper layers can apprehend more complex textures.

Classes that demonstrated constant amelioration with additional layers on both KITTI and Cityscapes for both metrics are *road, pavement, vehicle*. In the Cityscapes dataset, these categories corresponds to the most represented categories in terms of number of pixels while showing very little variation in appearance for *road* and *pavement*. This means that during training, these classes have been observed in a consistent manner and enough times for the additional layers to be beneficial.

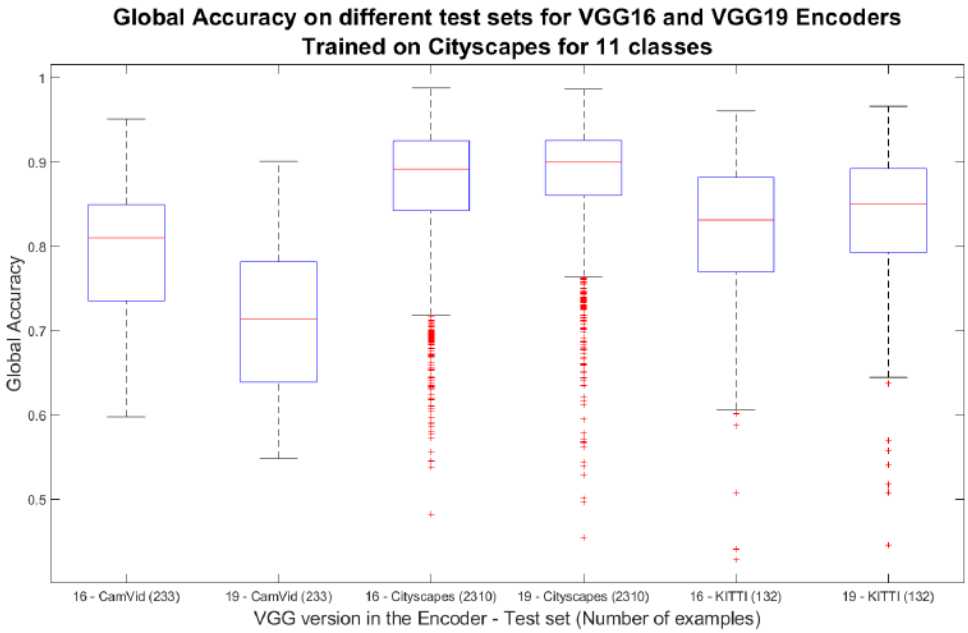


Figure 4.2: Comparison of Global Accuracy performance for SegNet with different encoders

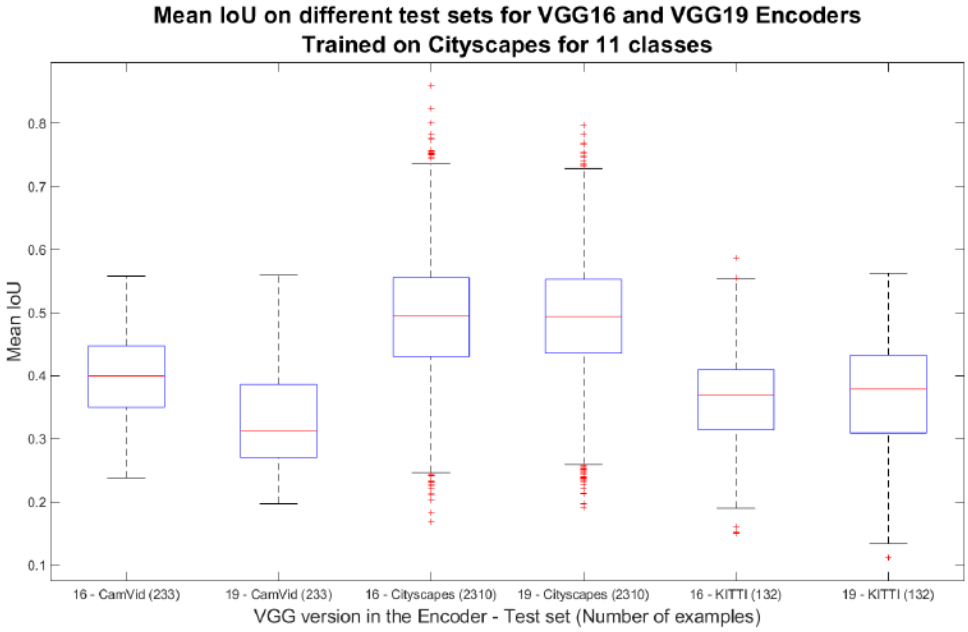


Figure 4.3: Comparison of mean Jaccard index performance for SegNet with different encoders

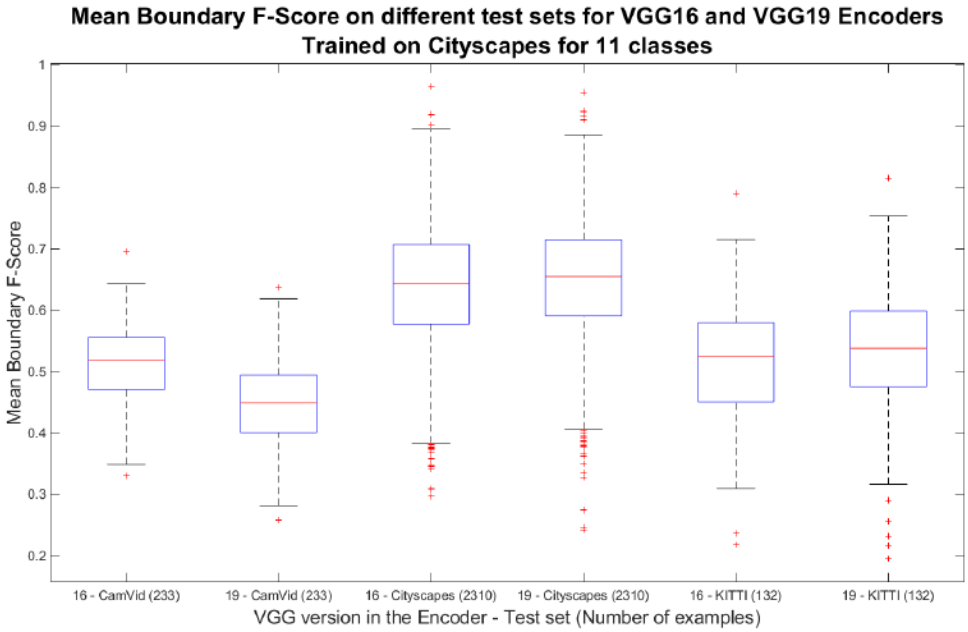


Figure 4.4: Comparison of mean boundary F-score performance for SegNet with different encoders

Overall, the addition of several convolutional layers does not yield an undeniable enhancement of the segmentation, thus I assess the results image per image. In Figure 4.2, Figure 4.3 and Figure 4.4, the image metrics for global accuracy, IoU and BF score respectively are shown as boxplots in order to illustrate the distribution of the results over the test sets.

All Figure 4.2, Figure 4.3 and Figure 4.4 display a significant amount of variability. This means that from an image to another, the quality of the segmentation is not reliable and consistent.

	Number of images predictions better with VGG19					
	KITTI (132)		CamVid (233)		Cityscapes (2,310)	
Global Accuracy	87	65.91%	30	12.88%	1223	52.94%
Mean Accuracy	74	56.06%	38	16.31%	1372	59.39%
Mean IoU	79	59.85%	24	10.30%	1169	50.61%
Weighted IoU	77	58.33%	36	15.45%	1326	57.40%
Mean BF score	87	65.91%	42	18.03%	1252	54.20%
Average improvement		61.21%		14.59%		54.91%

Table 4.5: Number of images showing a score improvement with SegNet VGG19

In Table 4.5, I quantify the number of images that actually showed a segmentation improvement according to the listed metrics. Even on Cityscapes, which was used for training, the segmentation improves for only one every two images. This suggests that the effect of the additional layers on multiple classification is here random.

In conclusion, the incorporation of convolutional layers by using the VGG19 encoder in SegNet on multiple classes semantic segmentation is not conclusive as there is no significant amelioration.

A possible reason is that the current architecture and training are not adequate to capture the increase in complexity in the network (multiple classification and added layers) and variety of examples in a large dataset of urban landscapes.

With a rise in number of parameters in the model of 10.6 millions, the training needs to be longer. Furthermore, adding 6 more layers might not be enough to observe a significant change. In a further work, libraries like YelloFin [95] could be used to

optimise the hyperparameters selection, including choosing the appropriate depth of the network.

4.4.2 Training validation

As the above results are not unequivocal, in this section, I look into the training and validation performance in order to determine the validity of the network's parameters. Recall that two versions of SegNet, VGG16 encoder and VGG19 encoder, have been trained on Cityscapes for 50 epochs. The trainings took respectively 22h and 26h to complete.

The accuracy and the loss for both training and validation are shown in Figure 4.5 for SegNet with the encoder VGG16 and in Figure 4.6 for SegNet with the encoder VGG19. Both trainings converged to a solution and do not show any sign of overfitting, that is to say that the validation measures stagnate and do not diverge.

However, training usually continues until the loss converges irrespective of the number of epochs, whereas I stopped all training at a specific epoch number. This means that the training for the deeper network suffered from a regularisation technique called early stopping. As a result, error analysis and disentangling factors of variations become difficult.

Assuming that an optimal error score is 0%, from training and validation performance in Figure 4.5 and in Figure 4.6, the networks exhibit some training error (about 11%) and slightly higher validation error (about 17%).

This suggests a high bias and a high variance. This means that the data is not understood as best as it could be by the networks and that the networks could generalise more accurately to unseen examples as well. In other words, the networks are underfitting the data.

Solutions to lower bias usually include developing a bigger network to increase its complexity, or training longer to improve its weights so that it fits the data better. Solutions to lower variance consist generally in using more data so that the network see more diverse examples in training or adding regularisation techniques to prevent overfitting.

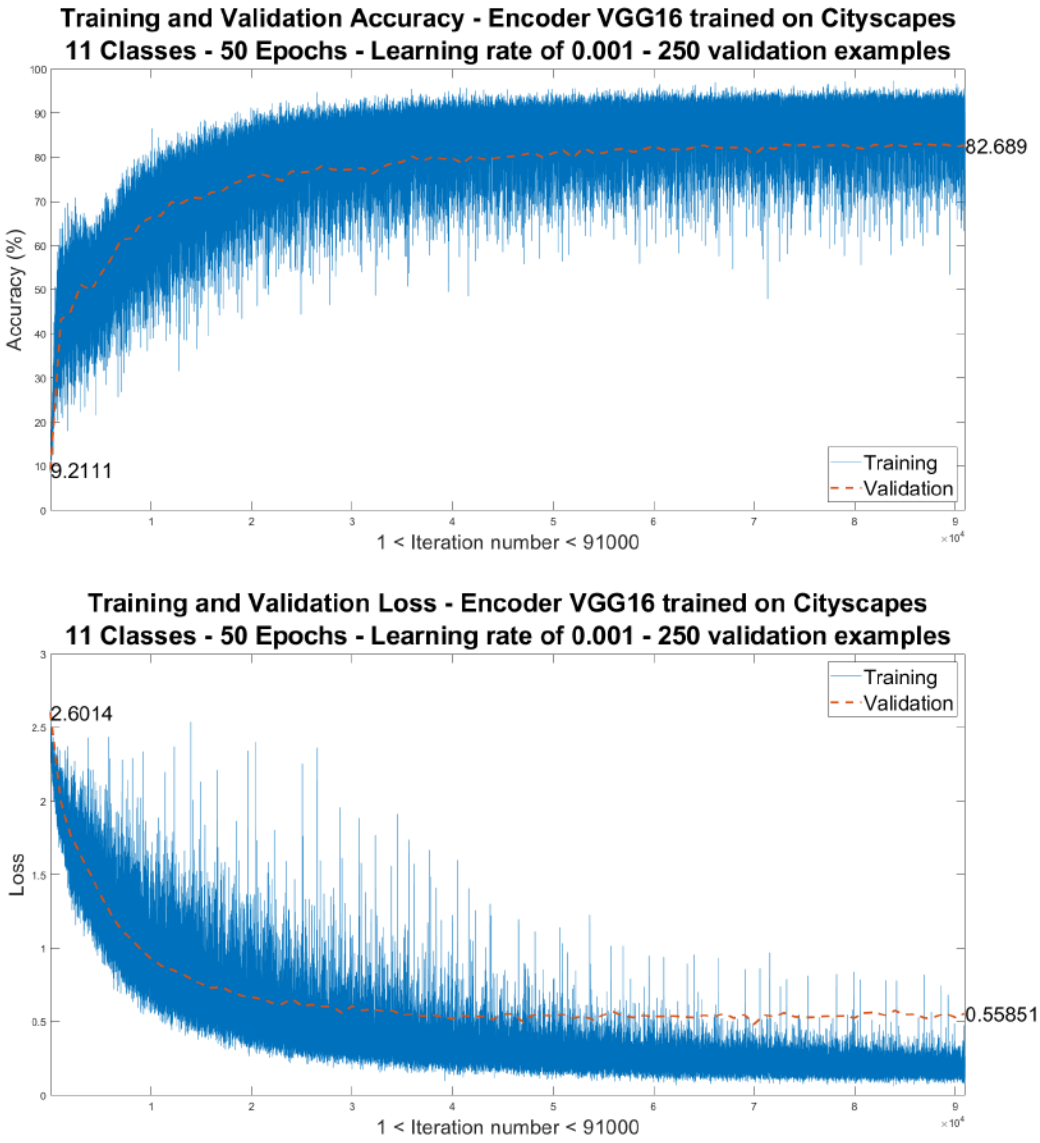


Figure 4.5: Training and validation performance for SegNet with encoder VGG16 - Accuracy above, loss below.

To conclude, early stopping on training as well as inconclusive results do not allow for a meaningful comparison between the two architectures on multiple classes semantic segmentation. The effect of additional layers cannot be fully appreciated and SegNet with VGG19 could show more potential with more training.

Considering the already significant size of SegNet with the VGG19 encoder (40 millions parameters as shown in Table 4.1) and the Cityscapes dataset containing more

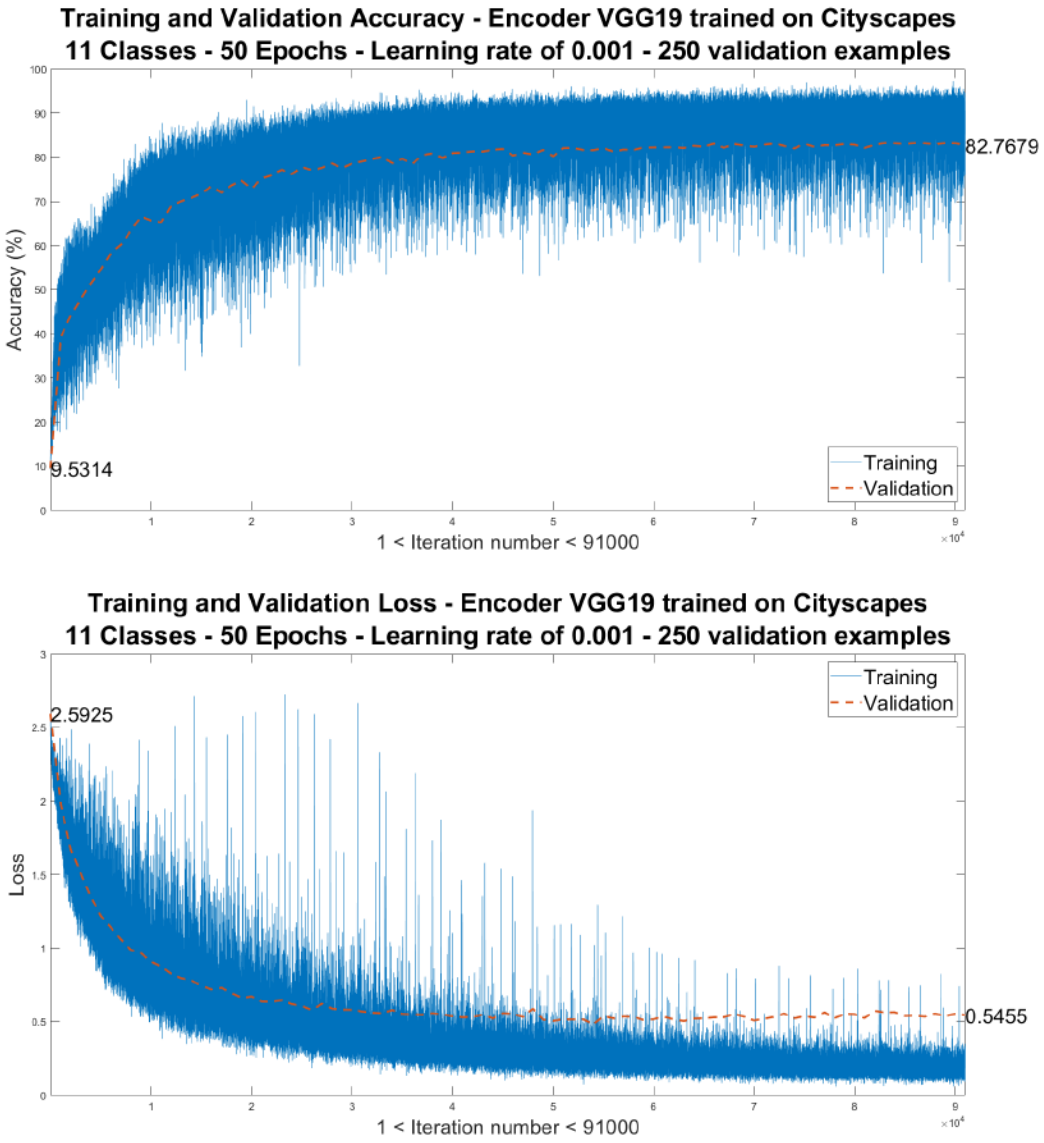


Figure 4.6: Training and validation performance for SegNet with encoder VGG19 - Accuracy above, loss below

than 25,000 examples, training longer is the best course of action for further work on multiple classification among the options listed above.

The assumption is made in the next section that reducing the complexity of the task will allow both architectures to be compared, and that a deeper network will yield a more precise segmentation.

4.5 Binary classification - vehicle detector

In this section, I present the results of the pixel-wise semantic segmentation on a binary classification problem. Specifically, I chose a vehicle detector, meaning that the network will only focus on detecting pixels belonging to the vehicle class. To do so, the classification layer is downsized from 11 possible outputs to two: *vehicle* and *background*.

Due to the nature of the task, there is a very large background class, which renders the interpretation of global metrics, such as global accuracy, very limited. The results are interpreted here based on the IoU and BF score only.

Firstly, I inspect the training and validation to confirm that no underfitting or overfitting problems occur. Secondly, I demonstrate that a deeper network (SegNet with the VGG19 encoder) improves the segmentation quality significantly, up to +22 pp in IoU and +32 pp in BF score.

4.5.1 Training validation

The training of SegNet with the encoder VGG16 took roughly 35.5 hours on a single GPU, and reached a final validation loss of 0.077 and a final validation accuracy of 97.96%. The training of SegNet with the encoder VGG19 took about 42.5 hours, reached a final validation loss of 0.078 and a final validation accuracy of 98.62%. Both graphs are in Figure 4.7.

Training and validation errors are both much closer to the Bayes optimal error (i.e. best possible error) with a training error close to 1% and 2% respectively for encoder 16, and below 1% and 1.4% respectively for encoder VGG19.

This shows a good balance with a low bias and a low variance in the network predictions. This means the network has understood the data well enough without overfitting and can generalise well to unseen images.

For the same training options as the multiple classification above and 25 epochs, it shows a training convergence much more appropriate and adequate for results evaluation.

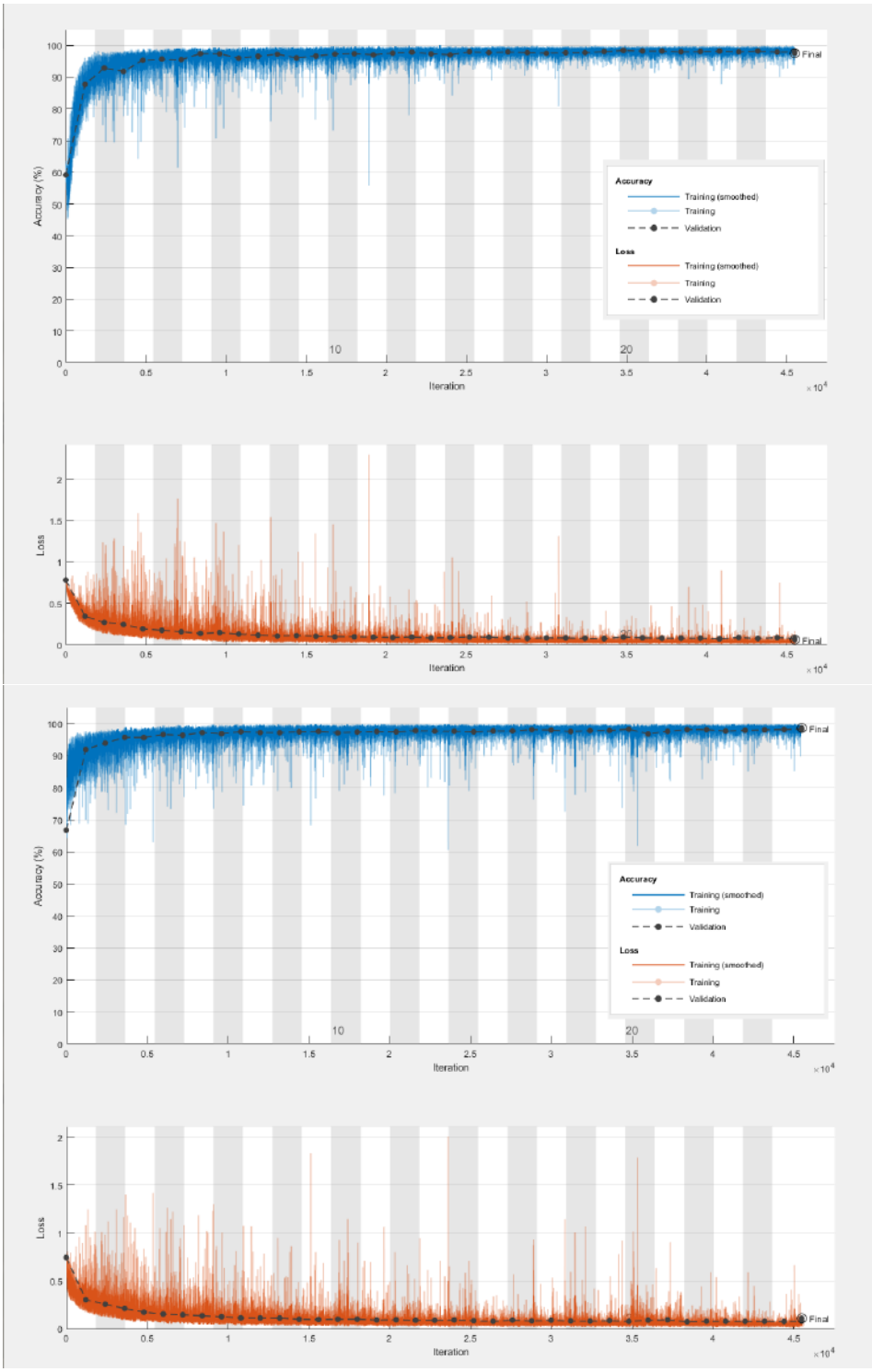


Figure 4.7: Training and validation performance for SegNet, with encoder VGG16 (above), with encoder VGG19 (below). Accuracy in blue, loss in orange.

	Trained on Cityscapes for 45,500 iterations VGG16 → VGG19 Tested on: (number of images in dataset)			Average improvement
	KITTI (132)	CamVid (233)	Cityscapes (2,310)	
Global Accuracy	96.143% → 97.641% +1.498	96.577% → 98.697% +2.120	97.676% → 98.543% +0.867	+1.495
Mean Accuracy	94.791% → 88.124% -6.667	93.951% → 89.483% -4.468	97.574% → 95.572% -2.002	-4.379
Mean IoU	77.777% → 82.521% +4.744	71.142% → 82.825% +11.683	86.682% → 90.470% +3.788	**+6.738**
Weighted IoU	92.062% → 93.905% +1.843	91.633% → 94.087% +2.454	92.742% → 94.086% +1.344	+1.880
Mean BF score	60.150% → 73.926% +13.776	44.357% → 61.383% +17.026	72.136% → 80.749% +8.613	**+13.138**
Class Accuracy	93.220% → 77.053% **-16.167**	91.075% → 79.388% -11.687	97.445% → 91.812% -5.633	-11.162
Class IoU	61.160% → 69.278% +8.118	48.690% → 70.485% **+21.795**	79.015% → 85.895% +6.880	**+12.264**
Class BF score	40.711% → 63.037% +22.326	28.090% → 60.112% **+32.022**	62.583% → 78.714% +16.131	**+23.493**

Table 4.6: Metrics results for binary classification - Vehicle detector

4.5.2 Results and discussion

The results of the *vehicle* semantic segmentation are shown in Table 4.6, where the metrics marked as “class” are for the *vehicle* class specifically.

The addition of the 6 layers in SegNet causes a small increase in global accuracy (+1.5 pp on average) but a remarkable increase in IoU and BF score. Indeed, the overall boundary precision improved by 13 pp, and by 23 pp for *vehicle* in particular, with up to + 32 pp on the CamVid dataset. The latter is equivalent to an astounding enhancement of 114% with the additional layers. The *vehicle* IoU also increased by a significant 12 pp overall.

This means that a deeper encoder in SegNet for a specific class detector results in a more accurate segmentation and contour definition. As observed in the examples in Figure 4.8, the segmentation is less noisy with fewer false detections and a more precise delineation for the correctly detected vehicles.

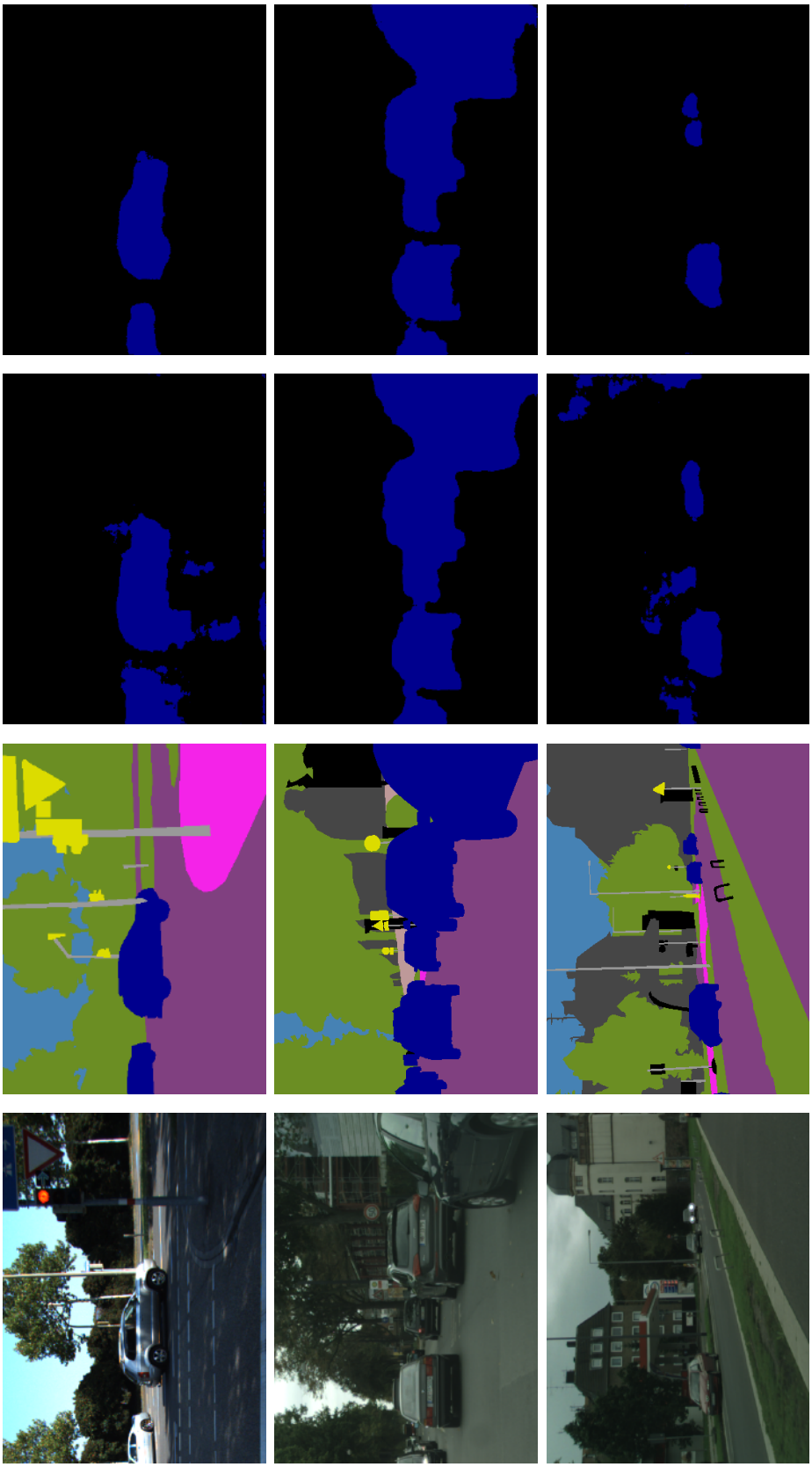


Figure 4.8: Segmentation results for binary classification - from left to right: left colour image, semantic ground truth, SegNet with VGG16 encoder, SegNet with VGG19 encoder.

		Intersection over Union: VGG16 ⟶ VGG19			
		Tested on: (number of images in dataset)			Average improvement
		KITTI (132)	CamVid (233)	Cityscapes (2,310)	
Trained on: (number of iterations)	KITTI (2,625)	56.335% ⟶ 63.321%	23.934% ⟶ 34.217%	52.399% ⟶ 57.752%	+7.541
		+6.986	+10.283	+5.353	
	CamVid (4,575)	58.966% ⟶ 64.218%	55.399% ⟶ 57.102%	54.040% ⟶ 65.614%	+6.176
		+5.252	+1.703	+11.574	
	Cityscapes (45,500)	61.160% ⟶ 69.278%	48.690% ⟶ 70.485%	79.015% ⟶ 85.895%	+12.264
		+8.118	**+21.795**	+6.880	

Table 4.7: Table Jaccard Index (IoU), for the *vehicle* class on binary classification

		Boundary F-score: VGG16 ⟶ VGG19			
		Tested on: (number of images in dataset)			Average improvement
		KITTI (132)	CamVid (233)	Cityscapes (2,310)	
Trained on: (number of iterations)	KITTI (2,625)	30.112% ⟶ 31.995%	13.047% ⟶ 14.811%	27.093% ⟶ 31.304%	+2.619
		+1.883	+1.764	+4.211	
	CamVid (4,575)	35.088% ⟶ 38.838%	29.427% ⟶ 30.335%	29.587% ⟶ 42.535%	+5.869
		+3.750	+0.908	+12.948	
	Cityscapes (45,500)	40.711% ⟶ 63.037%	28.090% ⟶ 60.112%	62.583% ⟶ 78.714%	+23.493
		+22.326	**+32.022**	+16.131	

Table 4.8: Table Boundary F-score, for the *vehicle* class on binary classification

In a CNN, the dimensionality reduction leads to a loss in spatial resolution in the feature maps as it loses boundary details in the image representation. The pooling indices in SegNet are keeping in memory the localisation in the image of the important features used for classification. This allows to store and retain this boundary information. Thus, there is a positive correlation between the BF score and the depth of SegNet.

In order to generalise these findings, following the same training procedure described in Section 4.2, I trained both architectures SegNet16 and SegNet19 on the three datasets separately: KITTI, CamVid and Cityscapes. Due to the smaller number of training examples in KITTI and CamVid, there are fewer training iterations for these.

I then cross-tested each model on the three datasets as well. The results are shown in Table 4.7 for the *vehicle* IoU and in Table 4.8 for the *vehicle* BF score.

Note that regardless of the amount of training received, there is a consistent improvement of the IoU and the BF score for the *vehicle* class by adding more layers in SegNet. However, the gain is largely more when the model is trained on a larger

dataset: +12 pp on IoU and +23 pp on BF score rather than only half and a fifth of that respectively.

Also note that the gain is generally more important when the network is tested on the datasets it has not been trained on. This is very interesting and significant as this means it is the generalisation to unseen and diverse examples that is also enhanced.

To conclude, the semantic segmentation task benefits from a deeper network with which boundary information is retained like in SegNet. There is also a benefit in reducing the complexity of the network and opting for a specific class detector rather than multiple objects classification. Artificial Intelligence is efficient to solve well defined and very specific problems. The broader the problem, the more it tends towards true intelligence rather than relying on statistic cues and mere computational power, which is obviously more complex to achieve.

Further work includes researching the optimum number of layers and adapting the loss or the training stopping criteria towards the metrics that are meaningful for the segmentation task. Indeed the cross-entropy checks accuracy pixel by pixel, so it primarily optimises the global accuracy.

4.6 Effect on the pipeline outputs

Recall that the semantic segmentation is used in my framework to isolate the vehicle points in the image, and then select them from the disparity maps to form the corresponding vehicle point cloud. In this section, I illustrate the effects of the experiments carried out in Section 4.4 and Section 4.5 on the point cloud formation and its aspect.

To do so, the SegNet predictions are inferred at original size (1024 by 2048), and passed on to the pipeline described in Section 3.2 up till step 3, the point cloud formation. The models of SegNet retained for this comparison are:

- SegNet with the encoder VGG16 on multiple objects classification, the *vehicle* class is extracted after,
- SegNet with the encoder VGG19 on binary classification for a vehicle detector.

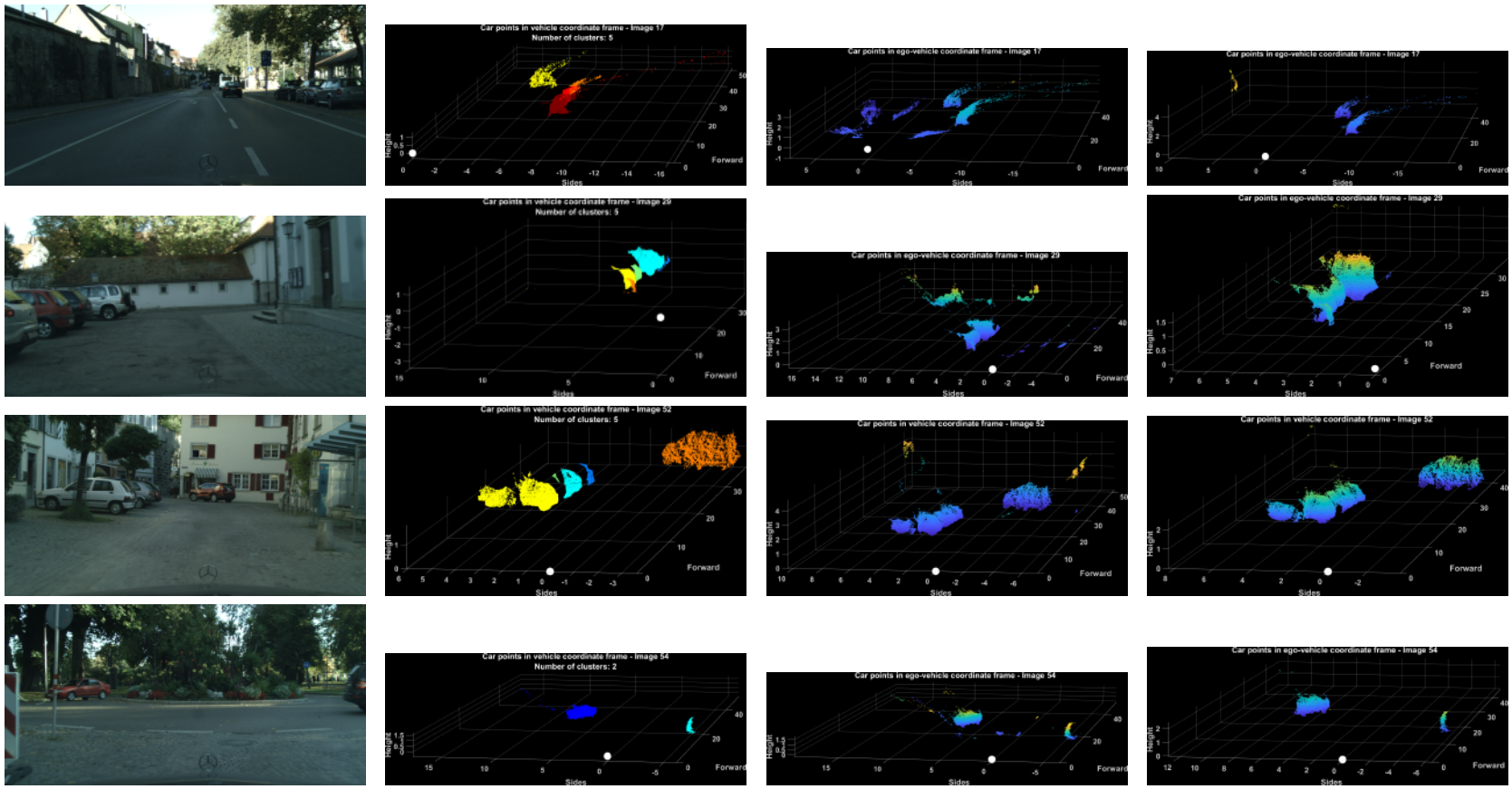


Figure 4.9: Effects of the segmentation on the vehicle point cloud.

From left to right: the original left RGB image, the corresponding target data, vehicle point cloud created with SegNet VGG16 on multiple objects classification, vehicle point cloud created with SegNet VGG19 for a vehicle detector.

The resulting vehicle point clouds are shown in Figure 4.9. The vehicle point clouds' aspect resembles more the target data point cloud when the segmentation is done for vehicle detection with SegNet19.

Improvements in segmentation helps reduce the false detections of vehicle points, sometimes entire wrongly detected vehicles, as shown in row 1 and 2. The gain in BF score helps removing scattering and discontinuities in the point clouds at the contours of vehicles, as observed in row 3 and 4.

To conclude, the semantic segmentation task needs to exhibit good performance in IoU and BF score to obtain a more realistic vehicle point cloud. Filtering out outliers by improving the semantic segmentation gives the best dispositions for the clustering algorithm in the following step.

This is the first time in the domain of autonomous driving that the importance of the BF score is highlighted. Indeed, in the literature [5], it is usually argued that boundary precision is not necessary for this application. The reason behind this argument is that outputs are often in the form of bounding boxes, with a large error margin applied for safety.

In future work, other methods could be tested for semantic segmentation, leveraging the most recent advances in benchmarks like the Cityscapes or KITTI on this task.

4.7 Summary

In this chapter, I investigated the first step of the pipeline I developed. The objective was to recognise information in an image, in particular classifying each pixel based on what it represents.

To do so, I tested a convolutional neural network designed to process images for recognition task. This network, SegNet, was developed for autonomous driving applications and trained for the semantic segmentation task.

I adapted the depth of the network by adding layers, and the complexity of the task by initially performing multiple objects classification and then binary classification - as a vehicle detector specifically.

My experiments demonstrated that when training with limited resources, favouring deeper networks on a simpler task resulted in a more accurate semantic segmentation with a significant increase in the precision of the boundary F-score (+23 pp on average).

In the context of the pipeline I developed, this enhancement in contour definition resulted in a vehicle point cloud containing less noise and thus its aspect resembles more that of the target data.

The choice of training the network to detect a specific class is not only supported by the encouraging results but makes complete sense regarding the philosophy of my proposed pipeline. Indeed, the final clustering stage can be applied to an individual class and thus serves the purpose of this thesis.

A sub-part of the above work led to a published poster at the Defence & Security Doctoral Symposium in 2018 (viewed by 2220, downloaded 1514) [96].

5. CLUSTERING

In Section 3.2, I presented a novel framework to identify and localise vehicles from short to mid-range distance [2m - 50 m] in a point cloud representation by using clustering methods on raw data. In this chapter, I explore different clustering techniques to identify each vehicle instance within the point cloud. Specifically, I discuss the advantages and limitations of two techniques:

- A distribution clustering method: Gaussian Mixture Models
- A density clustering method: DBSCAN.

I described in the previous chapter the role of the semantic segmentation step in detecting the correct vehicle pixels in the image and obtaining a decent point cloud to work with. In order to single out clustering methods considerations and provide a fair comparison analysis, I chose to make the experiment on the target data presented in Section 3.3.4. This will prevent inconsistency deriving from semantic segmentation to bias the interpretation of the clustering results.

The target data combines the disparity maps of the Cityscapes dataset with the semantic segmentation ground truth. The assumption is therefore made that the semantic segmentation step is ideal, with a perfect IoU and BF score. The vehicle point cloud is considered the closest possible to the reality, minding the accuracy of the disparity maps.

Challenges faced in this clustering step involve handling different sized vehicles as not only are encountered vehicles not alike but also distant and occluded vehicles in images yield fewer points in the point cloud. The angle of view (from the side, back, or front) and occlusion also modifies the shape of the vehicle.

As observed in Chapter 3, the nature of disparity maps adds on to the challenges because the variation of the disparity reduces with distance, resulting from the vanishing point problem where distant objects converge to a single point. This can be seen as a reduction of the resolution in distance estimation. Disparity maps computed from stereovision are also not immune to errors, notably due to reflective or transparent surfaces.

These challenges affect the data given to this clustering step and therefore render the identification of patterns, and thus vehicle instances, more difficult.

Although supervised clustering can be used to improve performance when labels are available, clustering is traditionally unsupervised learning so the aforementioned methods are unsupervised. Many of these techniques are based on a distance function or a similarity function in order to estimate how different pairs of examples in the data are. This means that the complexity of the algorithm, and thus its run-time, is proportional to the square of the number of points n, $O(n^2)$. Therefore, they rarely scale efficiently to millions of points and examples, as it is the case in autonomous driving datasets.

This is why clustering on raw data is not often seen in this application, even though DBSCAN has been used on LiDAR point clouds before [97]. Also, embeddings - i.e. representations - of the data are usually learned with neural networks to reduce the number of input features before applying a clustering algorithm [56].

As mentioned in Chapter 3, this work differs by creating a sparse point cloud from visual data, thus containing fewer points, and by not learning intermediate representations.

In this chapter, GMM and DBSCAN methods as well as the process for their parameter selection are described. The study and comparison of both methods demonstrate that GMM is, by design, not appropriate for this application, even though this is a very common method for clustering problems. Indeed, global co-variance matrices cannot fully encompass the diversity of the data features encountered in the scenarios.

On the contrary, DBSCAN presents suitable characteristics including noise identification. Furthermore, DBSCAN does not require to be given the number of clusters to find. This is particularly desirable here because the number of dynamic objects surrounding an autonomous vehicle varies and cannot be anticipated beforehand nor specified as a global parameter. Note that to my knowledge this is the first time that DBSCAN is applied to 3D environment representation data generated from visual information.

5.1 Distribution clustering

Distribution clustering identifies patterns in the data based on probabilities and how likely points are to belong to the same distribution. Recall from the pipeline overview in Figure 3.2 that the vehicle point cloud displays shapes that are not circular because only the sides, the front or the back of the vehicle are visible in an image. For this reason, a distribution clustering is well suited because the variance of the clusters can be adapted to fit an ellipsoidal shape rather than a circle. Among the most common approaches, Gaussian Mixture Model (GMM) works with Normal distributions.

Also, its complexity is $C * O(n)$, with C the number of clusters to be found, which makes GMM particularly attractive and practical for a point cloud of n points. In the following sections, I describe the method and justify the choice of co-variance matrix.

5.1.1 Gaussian mixture model

The principle of Gaussian Mixture Model is to fit a mix of Normal distributions to the data by maximising the probability that the sample points belong to the estimated Gaussians. Like many partitioning clustering algorithms, GMM requires to be given as input the number of clusters C to be found in the scene.

The probability that the i^{th} sample z_i is from the distribution k, $k \in [1 : C]$, is expressed by π_k in Equation (5.1) where μ and Σ are the mean and variance of the Normal distributions.

$$\pi_k = p(z_i = k|\mu, \Sigma) \tag{5.1}$$

The likelihood of observing a data point given that it came from the Gaussian k is:

$$\mathcal{N}(x_i|\mu_k, \Sigma_k) = p(x_i|z_i = k, \mu_k, \Sigma_k). \tag{5.2}$$

Combining both equations, the likelihood of observing the i^{th} sample taking into account all the distributions is the sum of all the likelihoods of observing the sample given it came from each possible Gaussian. This is expressed in Equation (5.3) for the i^{th} sample, and in Equation (5.4) for all N samples in the data.

$$p(x_i|\mu,\Sigma) = \sum_{k=1}^{C} p(x_i|z_i = k,\mu,\Sigma)p(z_i = k|\mu,\Sigma) \tag{5.3}$$

$$p(x|\mu,\Sigma) = \prod_{i=1}^{N}\sum_{i=1}^{C} \mathcal{N}(x_i|\mu_k,\Sigma_k)\pi_k \tag{5.4}$$

The paradox is that the parameters of each Gaussian distribution, the mean μ and the variance Σ, need to be known to cluster the data, but it is also necessary to know which samples belong to which distribution to estimate these parameters.

To do so, the iterative Expectation-Maximization (EM) algorithm is used, which initiate a partition of the data and optimise it until convergence by maximising the posterior probability that a data point belongs to its assigned cluster.

1. Initialise with random Gaussian parameters μ and Σ.
2. Repeat till convergence:
   - Expectation: compute the posterior probability $p(z_i = k|x_i,\mu_k,\Sigma_k)$, which is the probability that it came from the distribution k after being assigned to it.
   - Maximization: maximising the likelihood that each sample came from the distribution, $max(p(x|\mu,\Sigma))$.

By maximising the likelihood that each sample came from the distribution, the parameter of the Gaussians μ and Σ are updated to fit points assigned to them, which changes the aspect of the distributions. The EM iteration process can be seen as repeating an estimation of how confident the membership assignments are and then increasing the assignment confidence.

In my experiments, the initial conditions for the distributions parameters are random, and the maximum number of iterations for EM is set to 1,000. The number of clusters C that are expected to be found are provided by the target data. As GMM is here applied on a 3 dimensional point cloud, the Gaussian distributions are multivariate. As a consequence, different options are available for the co-variance matrix given in input to describe the variance within the data.

5.1.2 Co-variance options

In this section, the possible options for the co-variance matrices are explored. Co-variance measures the joint variability of two or more variables: in a multivariate Gaussian distribution, the variables can co-vary so the variances of each variables alone do not fully describe the distribution.

In statistics, co-variance is the mean value of the product of the deviations of the variables from their respective expected value, as expressed in Equation (5.5), where X are the variables. Equation (5.6), gives the variance of a single variable x.

$$cov[X_i, X_j] = \mathbb{E}[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])]. \tag{5.5}$$

$$\Sigma(x) = \mathbb{E}[(x - \mathbb{E}[x])(x - \mathbb{E}[x])] = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2 = \sigma^2(x). \tag{5.6}$$

The type of co-variance matrix specified for GMM changes the distributions GMM fits on the data. The possible settings for the co-variance matrix are: either diagonal or full, and whether all clusters should share the same co-variance matrix or not. This setting therefore affect the shape and the size of the distributions searched for by GMM as it changes the matrix initialisation and constrains the maximization step in EM.

Geometrically, the co-variance matrix determines the shape and orientation of the distribution ellipsoid over the cluster, which is illustrated in Figure 5.1. A diagonal co-variance matrix contains only the variances of each parameter, which signifies the variables do not co-vary and are linearly uncorrelated; the distribution spread is aligned with the axis of the feature space. On the contrary, a full co-variance matrix means the variables co-vary and are somehow correlated, with the sign indicating the correlation; for example, as x decreases, y increases.

A shared co-variance would be assuming that all the clusters, i.e. vehicles, have the same variance in rows, columns and disparities. Thus, vehicles would all appear the same way and with the same size.

In order not to assume a particular behaviour on the target data for 3D panoptic

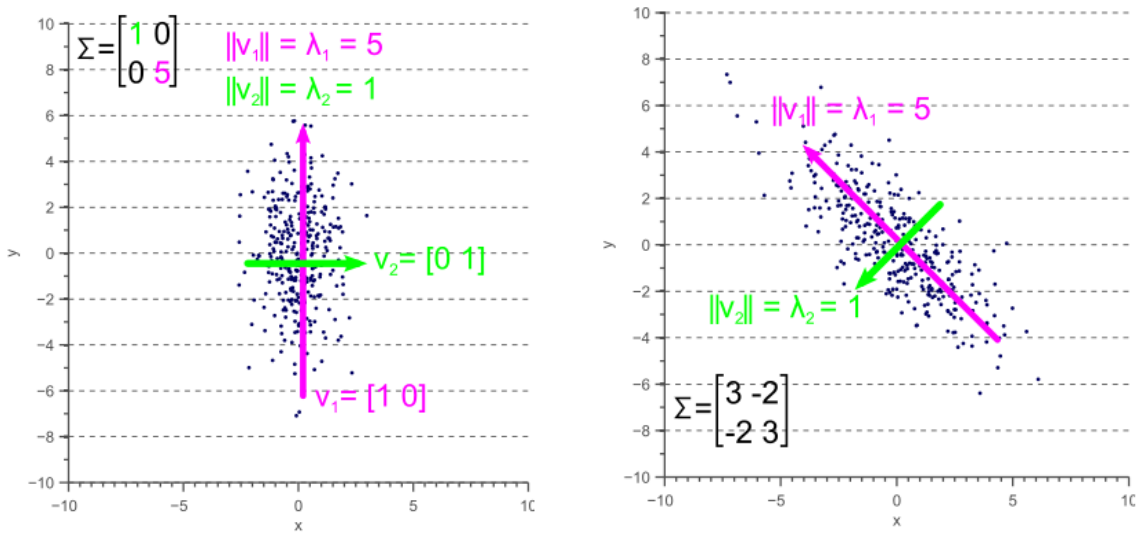


Figure 5.1: Geometrical interpretation of the co-variance matrix: diagonal (left) and full (right). Reproduced from [98]

segmentation based on visual data, all co-variances possibilities are tested in the experiment:

- Diagonal and shared co-variance (DS),
- Diagonal and not shared co-variance (DNS),
- Full and shared co-variance (FS),
- Full and not shared co-variance (FNS).

The results are presented in Section 5.3, along with the comparison with DBSCAN, which is described in the following section.

5.2 Density clustering

In density-based clustering methods, also called **spatial** clustering, clusters are recognised as areas with high density of points separated by areas with lower density. These areas with lower density can either be noise or outside the cluster. Detecting such clusters is therefore entirely based on and assuming density variations within the data.

This assumption is particularly well suited to the sparse nature of the point cloud

from the *vehicle* class, created within my framework. Indeed, in a driving scenario, unless vehicles have crashed into each other, entities are naturally separated and thus satisfy this assumption. This is the main reason why working with 3D classified point cloud especially for dynamic objects is much more advantageous than staying in a 2D image plan domain that do not encompasses occlusion problems.

In the next sections, I describe the approach and parameters selection for DBSCAN.

5.2.1 Density-based spatial clustering of application with noise

The notion of density is formally defined for the first time by DBSCAN [99]. In order to identify a cluster, DBSCAN uses connectivity between each point. If for a point, its neighbourhood of a given radius contains a minimum number of other points, then the density exceeds the required threshold for the point to be considered part of a cluster.

The author of DBSCAN formalises the definition of the neighbourhood of a point p, noted $N_\varepsilon(p)$ by Equation (5.7), where q are the points from the database D that are at a distance, as computed by the function $dist$, smaller than the given radius ε.

$$N_\varepsilon(p) = \{q \in D | dist(p,q) \leq \varepsilon\} \tag{5.7}$$

With DBSCAN approach, a cluster is defined by the highly dense points inside of the cluster, called the core points, and the points on the border. As the ε-neighbourhood of border points is less dense than those of core points, naively following the sole definition above would require to set the minimum number of points too low in order to encompass all the points belonging to the cluster.

In order to retain a meaningful parameter setup, the notions of *directly density-reachable*, *density-reachable*, and *density-connected* are introduced in [99].

Directly density-reachable means that "for every point p in a cluster C, there is a point q in C so that p is inside of the ε-neighbourhood of q and $N_\varepsilon(q)$ contains at least" the minimum number of points. The effect is to include the border points, by being close to a core point, i.e. directly density-reachable from the core point.

Following these notions, a cluster is defined as a set of points that are connected to

each other, including and maximised by all density-reachable points; while noise is the rest of the points. More formally, as per [99]:

“

A *cluster* C wrt. ε and $MinPts$ is a non-empty subset of D, [the database of points], satisfying the following conditions:

1. $\forall p,q$: if $p \in C$ and q is density-reachable from p wrt. ε and $MinPts$, then $q \in C$. (Maximality)
2. $\forall p,q \in C$: p is density-connected to q wrt. ε and $MinPts$. (Connectivity)

Let $C_1,...,C_k$ be the clusters of the database D wrt. parameters ε_i and $MinPts_i$, $i = 1,...,k$. Then we define the *noise* as the set of points in the database D not belonging to any cluster C_i, i.e. $noise = \{p \in D | \forall i : p \notin C_i\}$

”

The parameters ε and $MinPts$ are global parameters and can be set with a heuristic to the least dense cluster in the database that would not be considered as noise. In the current autonomous driving application based on visual data, the least dense cluster is the foreground object. The selection of ε and $MinPts$ are outlined in the next section.

The algorithm is a two-step approach: firstly, pick a random point p, then find all the density reachable points wrt. ε and $MinPts$. If p is a core point, a cluster is created. If p belongs to the border or is noise, then the density around p is low. Thus, there will not be enough points in the ε-neighbourhood, meaning that no other point is density-reachable from p. DBSCAN will thus pick another random point.

5.2.2 Parameter selection

Minimum number of points

The choice of $MinPts$ relies on domain knowledge. In this work, each point in the data initially corresponds to a pixel of the image. The question is therefore: which is the minimum number of pixels required to form a cluster? More specifically in the

case study: how many pixels are required to form a vehicle?

According to the author, $MinPts$ must be strictly larger than the number of dimension of the feature space. As DBSCAN is applied onto 3D points, with no other features, that minimum here would be four. However, so few pixels represent either a largely occluded vehicle or, a very distant vehicle. In both cases, these are rare occurrences so it does not provide a realistic value of the actual minimum encountered.

More importantly, perspective makes the minimum vary across the data while DBSCAN assumes similarly dense groups. As the accuracy of the depth estimated by stereovision drops in the background, only the first 50 m are used.

By following the projective geometry principles introduced in Chapter 2, a range for the minimum number of vehicle pixels can be estimated for an average car size between 1.7 m and 2 m wide. It is likely to appear the smallest at 50 m, which corresponds to a width of 77 to 91 pixels on the image. This magnitude is corroborated by parameter examples for the same application but with LiDAR data. Empirically, I adjust $MinPts$ to 80.

Neighbourhood radius

Once $MinPts$ is set, the size and the shape of the neighbourhood around the point in which to search for that minimum number of points can be estimated. The shape of the neighbourhood can be changed by modifying the distance function used to compute the point to point distance. The Euclidean distance is used, so the neighbourhood search area is a sphere.

A common method to determine the value of the radius, ε, is to compute a k-distance graph. For each point in the data, the distance to the k nearest points is computed and then sorted by magnitude.

The resulting graph generally forms an arm with two distinct parts, as shown in Figure 5.2. The left part, relatively flat or with a slow increase, are points that have similar distance to each other and can be thus considered as a dense group, i.e a cluster. At the right end of the graph, the curve exhibits a significant increase and diverges. The distance to these points is significantly larger than for the rest of the points and therefore they are unlikely to be part of the cluster.

The goal is to determine the distance value at the elbow of that arm as it corresponds to the distance threshold from which points stop belonging to the group. Because the value of k is taken to be the parameter $MinPts$, the distance threshold found is a good estimation of the radius ε for the minimum number of points considered.

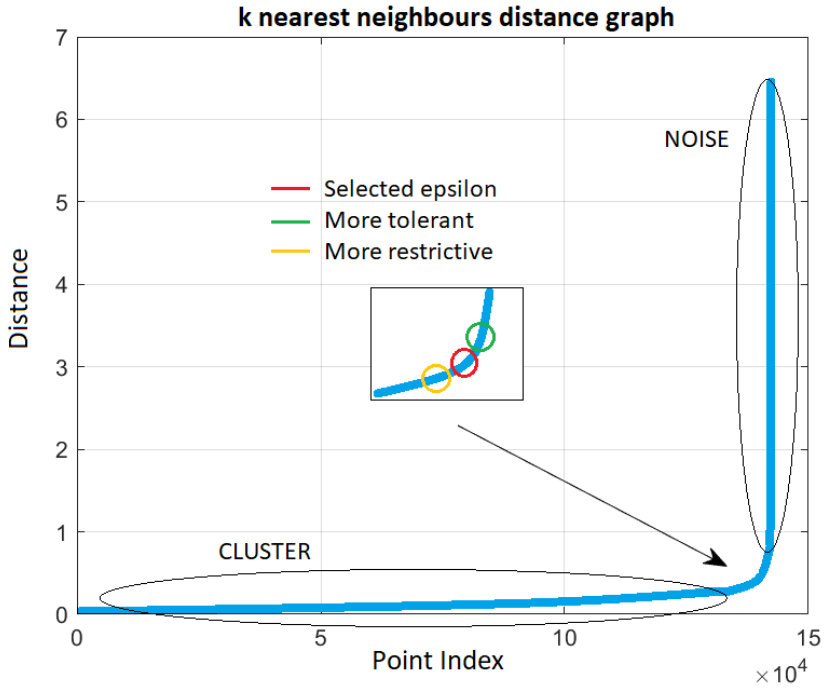


Figure 5.2: Epsilon selection on a k-distance graph.

It is interesting to note here that this "elbow" is not necessarily a sharp corner, but resembles more a curve. This means that the selection of ε can be more or less conservative depending on where on this curve the threshold is selected. Eventually, this affects the delimitation of the **border** points of the clusters and, thus the **noise** points identified by DBSCAN.

When a single estimation is sufficient, ε can be manually selected. However, in the event of large datasets, repeatedly selecting epsilon manually is firstly, impractical and time-consuming, and secondly, it can introduce variability.

In this work, this process must be automated as an estimation is necessary for each image. Estimating the radius ε automatically is the topic of much research [97] [100] [101]. I use a simple geometric approach to solve this problem, as shown in Figure 5.3, inspired from MATLAB.

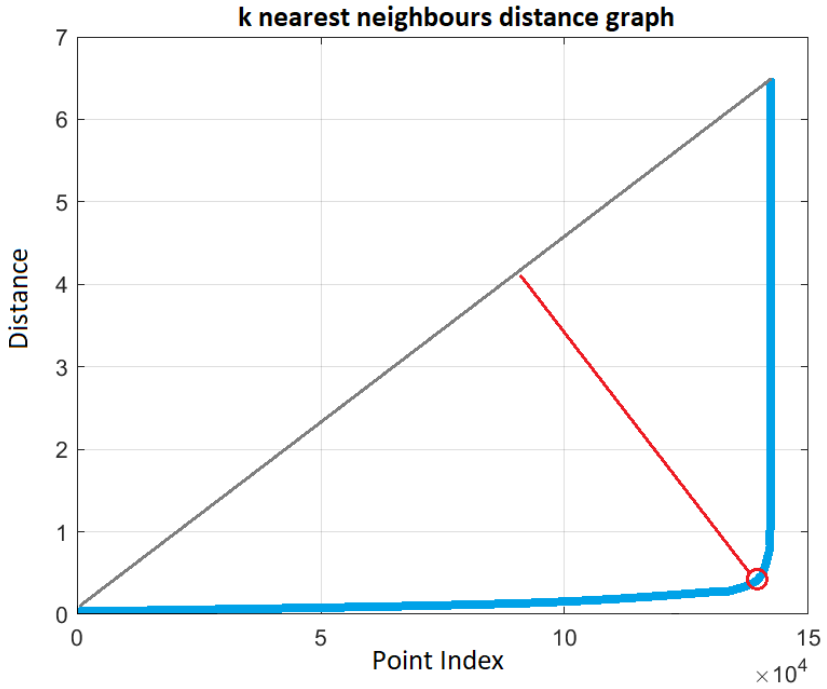


Figure 5.3: Automation of ε selection.

It considers the line created by both extremities of the k-distance graph (in grey) and find the point the furthest away from it (red circle). The distance of a point (x_0, y_0) from the line defined by two points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ is expressed in Equation (5.9). The distance value, i.e y_0, for the point maximising the distance from the line is selected as ε. In the example given in the Figures, ε is 0.55.

$$\varepsilon = \arg\max_{y}(d(P_1, P_2, (x, y))) \tag{5.8}$$

$$d(P_1, P_2, (x_0, y_0)) = \frac{|(y_2 - y_1)x_0 - (x_2 - x_1)y_0 + x_2 y_1 - y_2 x_1|}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}. \tag{5.9}$$

Note that MATLAB has two implementations of DBSCAN algorithm: one in the Phase Array System Toolbox and a different one in the Statistics and Machine Learning Toolbox. The former one contains more options such as the automatic selection of ε and handling density variations, while the latter, surprisingly, does not.

Unfortunately, the latter is optimised to handle large amount of data, while the im-

plementation developed for radar data is not. As a result, MATLAB source code constantly generates memory errors when computing the k-distance graph for ε selection. Its upper limit, dependent on the 16GB of RAM used in this work, is around 40,000 points, which equates to a region of interest of 200 by 200 pixels in the image. For information, considering my dataset and cameras, a van at a distance of 20 m can produce a region of interest of 340 by 320 pixels; it is only one vehicle but it is already too much data to process.

Thus, as MATLAB implementation for an automatic ε selection does not suit my data, I re-implemented it with the Statistics and Machine Learning Toolbox. The pipeline can now handle hundreds thousands of points. Also, implementing this automatic selection reduced the computation time by half compared to manual selection: from 1h and 1h30 to 30 and 40 minutes respectively for two experiments made on the 59 images recorded in Lindau.

5.3 Results and discussion

In this section, I compare GMM and DBSCAN for *vehicle* instance clustering on a 3D point cloud created from visual data. I demonstrate that GMM is limited by its parameter settings that are on a case by case basis and therefore cannot be generalised to all the scenarios encountered in an autonomous driving application. In contrast, DBSCAN is more flexible.

In Section 5.1.2, four co-variance options were described for GMM, and their effect on the clustering of the point cloud is shown in Figure 5.4 and Figure 5.5. The top rows show the distributions fit with diagonal matrices, while in the bottom row the co-variance matrices are full; the left column uses shared co-variance matrices while the right column show distributions without shared co-variances.

In Figure 5.4, there are two vehicles, both seen from the back with one vehicle appearing smaller because it is more distant. Among all the co-variance options, the best fit observed for this scenario is obtained with a diagonal co-variance which is not shared between the two clusters.

Indeed, as the vehicles do not have the same size, and are not at the same distance, their variance in rows and columns differ; thus, each vehicle needs a different co-

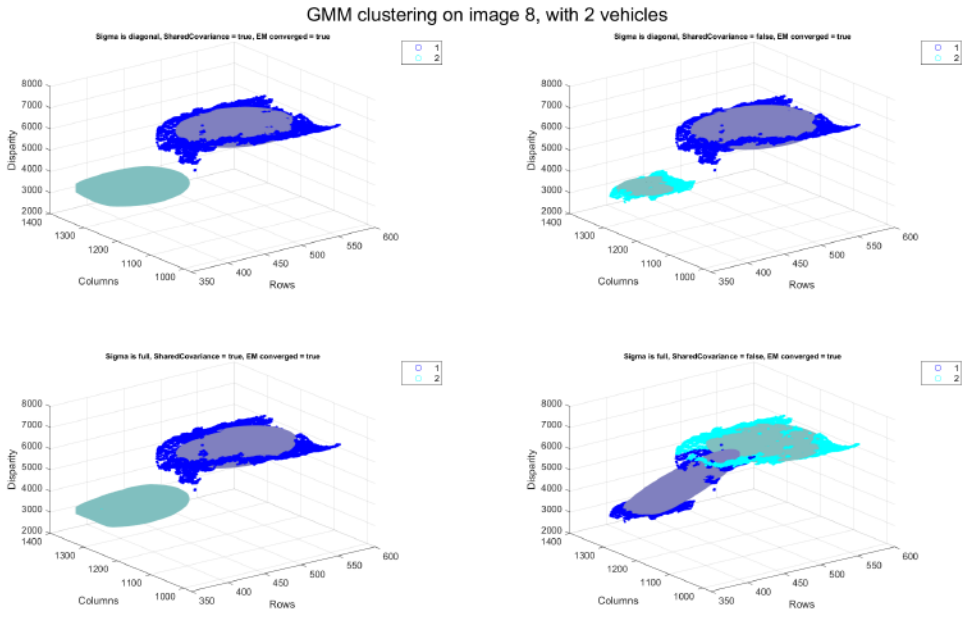


Figure 5.4: co-variances options. Scenario 8.

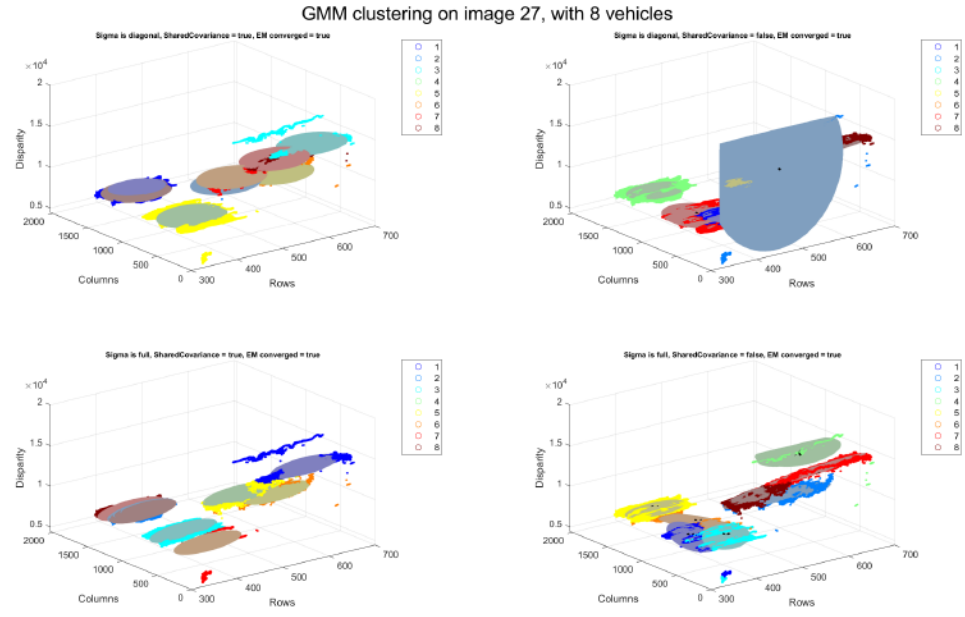


Figure 5.5: co-variances options. Scenario 27.

variance matrix. Additionally, the spread of the points is fairly even along rows and columns and do not vary in disparity. This is because only the back of the vehicle is seen, perpendicular from the camera's point of view. This explains why a diagonal is sufficient to capture the distribution of the vehicle.

In Figure 5.5 however, the situation is not as straightforward with more vehicles. With diagonal or full matrices that are not shared between clusters, the distributions fit found by GMM are somehow incoherent spatially. These two scenarios suggest that the co-variance matrix option cannot be set as a global parameter, as the best fits for both scenarios are not obtained with the same co-variance option.

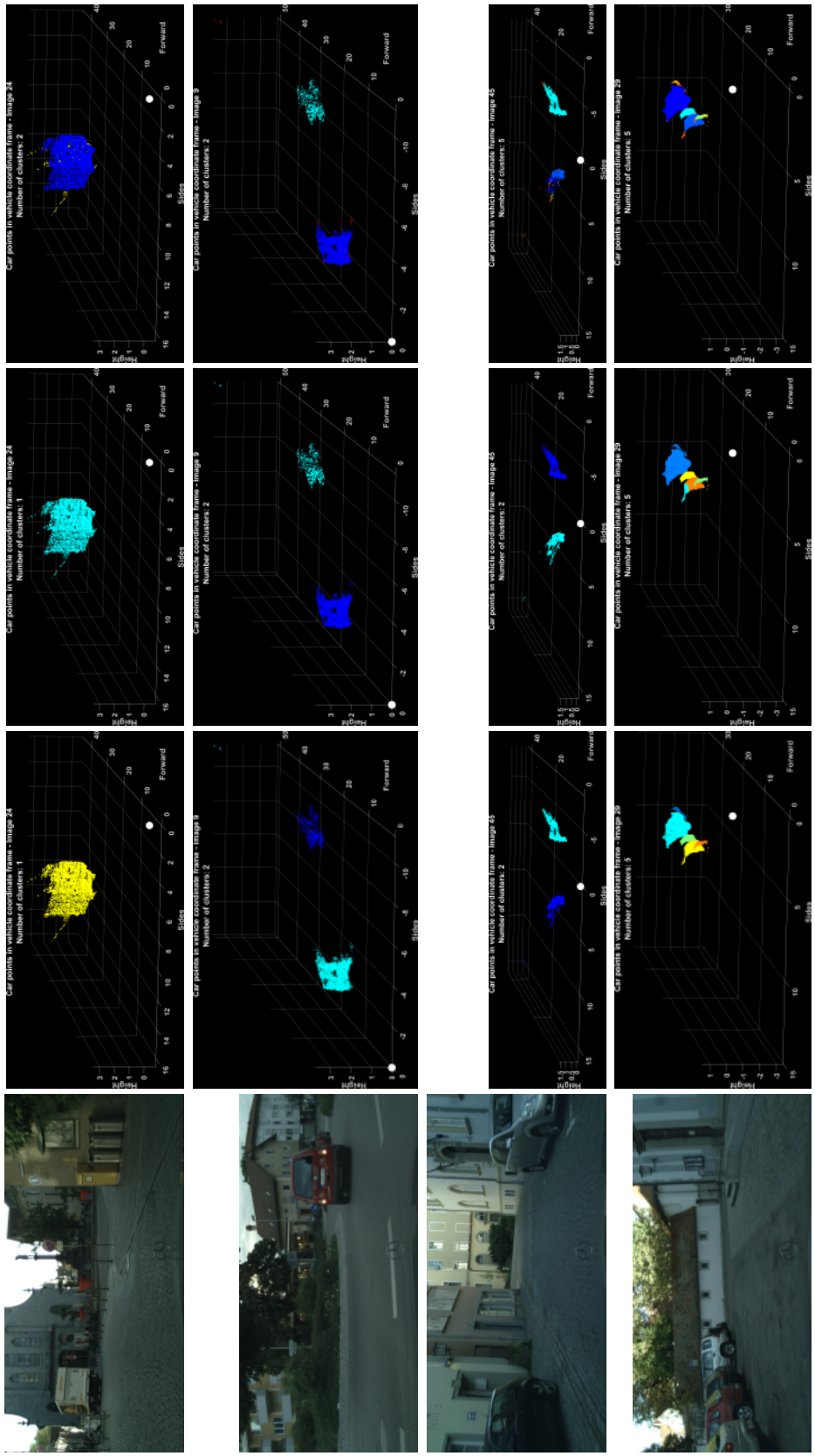
For the rest of the section, the co-variance matrix was set to a diagonal and not shared option. Not all vehicles are at the same location, they are different in sizes and in shapes. In the point clouds, one can observe that the vehicles appear relatively "flat", so there is not a lot of variation in disparity values and variables do not appear correlated. For this reason, the co-variance matrix was set to diagonal.

The comparison between GMM and DBSCAN is shown on a selection of examples in Figure 5.6, where GMM and DBSCAN perform similarly, and in Figure 5.7, where DBSCAN outperforms GMM thanks to its flexibility.

In Figure 5.6, row 1 and 2 are relatively simple scenarios with only one and two vehicles respectively. GMM identifies the instances correctly in both; however, when there is one vehicle, GMM simply estimates the variance of the whole point cloud. Also note that DBSCAN identifies noise in the surroundings of the instances which results in more coherently sized-vehicles.

In row 3 of Figure 5.6, GMM separates the two vehicles correctly. Recall from Section 3.4.2 that the same example led to clusters being divided top/bottom instead of left/right. The EM algorithm is sensitive to initial conditions and it is likely to have played a role in this case. On the other hand, while DBSCAN detects both instances as well, it also identifies three false detection due to the scattering in the point cloud.

In row 4, the scenario is relatively complex with five vehicles parked on the left among which the fifth and furthest vehicle is barely visible. DBSCAN performs notably well on this case, identifying that last vehicle and surrounding noise too. Conversely, while GMM detects four of the vehicles, the last one is not detected.



From left to right: left image, target point cloud, clusters by GMM, clusters by DBSCAN.

Figure 5.6: Comparison of GMM and DBSCAN on the target data. scenarios 24, 9, 45 and 29

As a reminder, it is important to note that GMM is given the initial number of clusters to look for. Consequently, in this particular example, it does not adapt and instead, it forces the algorithm to divide another correctly detected vehicle into two instances.

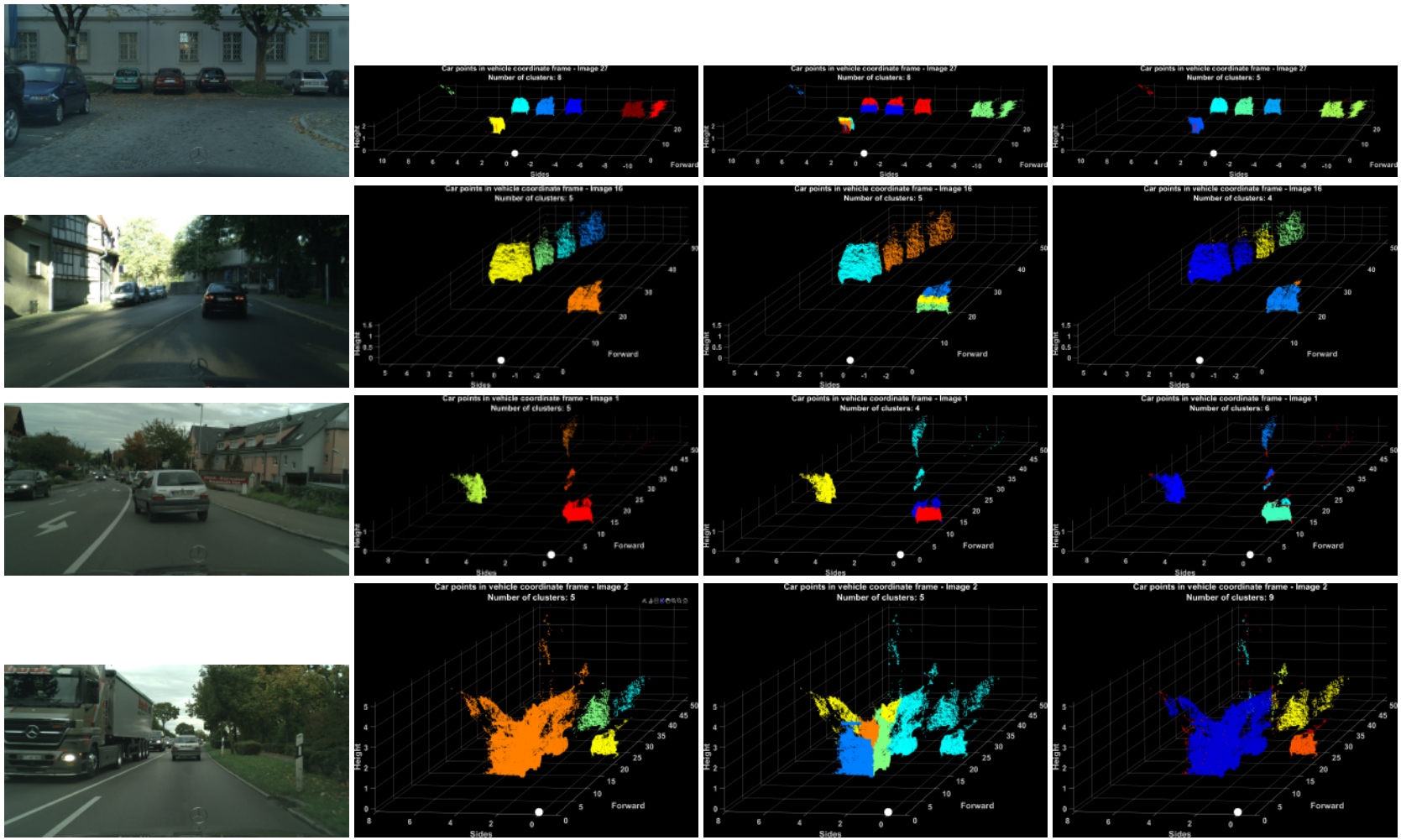
In Figure 5.7, the examples demonstrate the limitations of a distribution clustering technique for a spatial application. Results clearly show that DBSCAN is more suited to this application and thus outperforms GMM.

In row 1, 2 and 3, GMM exhibits the same behaviour by grouping the background vehicles as one instance but dividing one foreground vehicle into several clusters. It is important to remember that the resolution in disparity significantly reduces with distance, so the variance in disparity for distant vehicles is very small even if they are separated. Despite the co-variance matrices not being shared between clusters, GMM fails to recognise too distinct distributions. On the contrary, DBSCAN identifies more vehicles and noise, while GMM stays sensitive to outliers.

In row 4, the scenario is comprised of a truck on the opposite lane and three other vehicles. GMM fails to identify any one of them correctly for two reasons, one being the same as for the other examples. More importantly, the second reason is that the co-variance matrix cannot encompass the distribution of this truck.

Indeed, regardless of the co-variance matrix option on this example, the front and the side of the truck are constantly separated and never grouped as one vehicle. As shown in Figure 5.4, the front, or the back, of a vehicle display a relatively low variance in disparity when they are parallel to the image plane; this is particularly well captured by a diagonal co-variance matrix.

However, the side of the truck is affected by perspective, and in this case, there is some co-variance between the disparity values and the pixel locations. Therefore, not only the co-variance is image-dependent, but it is also vehicle-dependent, and it cannot fully describe the shapes of large vehicles. As a consequence, and adding that GMM cannot estimate the number of cluster automatically, GMM cannot scale well to the diversity of the scenarios encountered in autonomous driving.



From left to right: left image, target point cloud, clusters by GMM, clusters by DBSCAN.

Figure 5.7: Comparison of GMM and DBSCAN on the target data. scenarios 27, 16, 1 and 2

On the contrary, in row 4, DBSCAN correctly identifies the truck and the vehicle in front of the ego-vehicle. It fails though to separate the two vehicles in the background, and the scattering in the point cloud coming from inaccuracies in the depth estimation leads to multiple false detection.

It is worth noting that several additional actions could be followed to tune the GMM's behaviour on this type of point cloud for 3D panoptic segmentation and improve its performance. These actions include changing the initial conditions from random to a pre-estimation of the clusters, with either k-means or another technique.

Also, in this case, the features would benefit from a zero-mean normalisation and from using information criterion measures to adjust the number of cluster when necessary. However, these actions may not be straightforward to implement and would not ease generalisation of the approach.

However, DBSCAN shows more potential on spatial point clouds as it copes with several of the limitations experienced by GMM; it identifies outliers as noise, discovers clusters with arbitrary shapes (not necessarily ellipsoids), handles more complex examples with more vehicles, determines the number of clusters automatically and thus requires minimum domain knowledge to set its parameters.

As DBSCAN is based on density variations though, it is also affected by distant objects due to the disparity resolution decrease. The issue in using a global parameter setting for ε is that DBSCAN can merge two clusters into one if two clusters are too "close" to each other, separated by less than ε. These clusters could be of different densities, but if they are both denser than the threshold defined by the global parameter then they will be merged, irrespective of a density variation within the cluster. This is illustrated in Figure 5.7, row 1, 2 and 4, and addressed further in Section 5.4.1.

That being said, in a navigation context it is worth mentioning that this issue is not critical as the most important objects to be detected are the ones in the immediate surrounding of the ego-vehicle. These are well detected and clustered using DBSCAN. Distant objects will eventually, either disappear or come closer, which would be resolved as DBSCAN deals pretty well with close by objects.

5.4 Going further with DBSCAN

5.4.1 Feature space: RCD vs XYZ

In this section, I explore the features space in which the clustering is performed. Following the description of my pipeline in Section 3.2, I defined the features space as the dimensions in which the point cloud is projected before performing the step 4, clustering. For example, in its original version as presented in Section 3.2, these dimensions are *Rows, Columns,* and *Disparity values*, abbreviated as *RCD* features space.

Theoretically though, the step 5, mapping, can either happen before or after the clustering. This imply that the clustering step could actually be executed after mapping on the *XYZ* features space, which is the ego-vehicle coordinates frame. This is actually done in the literature when the point cloud is captured from LiDAR.

As clustering techniques are usually sensitive to the input features, I tested the two alternatives with DBSCAN:

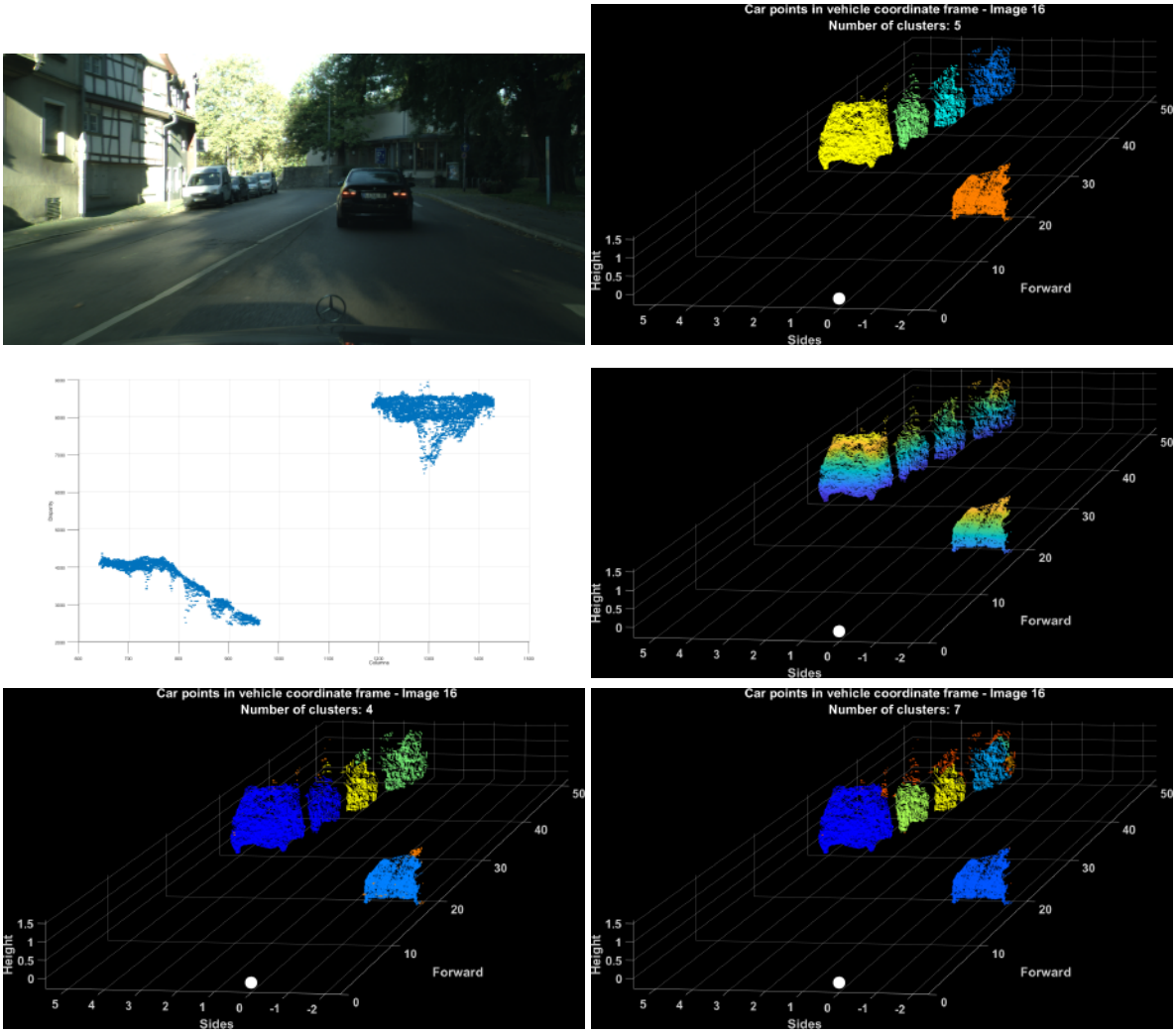
- Step 1, 2, 3, then clustering (4) on RCD space followed by mapping (5).
- Step 1, 2, 3, then mapping (5), followed by clustering (4) on XYZ space.

Contrary to a LiDAR point cloud that is formed from direct measurements, generating a point cloud from visual data derives from indirect measures of depth that implies some levels of image interpretation through feature detection, matching and estimating a disparity map.

Furthermore, beyond a certain distance, the disparity resolution is too small. This means that for a fixed distance interval, the disparity variation reduces with distance. This can be seen in Figure 5.8, middle left image, where one car spans over 1000 disparity values in the foreground, whereas a similar sized-car in the background spans over 500.

As a consequence of this disparity inaccuracy, and because the distance is inversely proportional to the disparity, the scattering once projected in real world coordinates will become spatially inconsistent and will stretch toward infinity as the distance increases.

Hence, unlike LiDAR data which is relatively uniform in accuracy, point cloud from visual data suffer from two issues: firstly, variation in resolution and secondly, inaccuracy increases with distance.



Top left: Left image. Top right: Target point cloud. Middle left: points in RCD features space. Middle right: points in XYZ feature space. Bottom left: Clusters by DBSCAN run on RCD feature space. Bottom right: Clusters by DBSCAN run on XYZ feature space.

Figure 5.8: Feature space comparison.

As a result, the behaviour of DBSCAN is dependent on which feature space it is performed. This is because DBSCAN is sensitive to variations as its neighbourhood radius ε is set as a constant through the point cloud.

Two things can happen, both illustrated in the bottom row of Figure 5.8:

- when executed on RCD space, the reduction of the disparity resolution can lead to two clusters being too close to each other and appear connected. DBSCAN will therefore group the two together and not recognise that they are two separate entities, like the dark blue vehicles in the left point cloud.
- when executed on XYZ space, the scattering in the background lead to points further apart than they should be. DBSCAN will thus not consider them connected and one vehicle can be divided in several clusters, like the back vehicle divided in 3 clusters in the right point cloud.

For both occurrences, further work includes the usage of OPTICS [102]. OPTICS is an upgrade on DBSCAN that can identify the internal structure of the data and overcome the setting of global parameters. Therefore, the neighbour radius ε could potentially adapt itself to discover the clusters appropriately. The MATLAB implementation I attempted suffered from memory management issues and could not handle the numerous points of this autonomous driving application.

5.4.2 Epsilon is linked to the disparity

While OPTICS could assist with the setting of the neighbourhood radius ε, it does not define the concept of noise as well as DBSCAN. Moreover, in the current context where the point cloud is created based on the projection from an image, the neighbourhood radius ε can be linked to the camera set-up.

I describe in this section how to link ε with disparity. I then verify numerically the values of the neighbourhood radius ε.

In the previous section, I defined two features space:

- RCD, the point cloud in the image coordinate frame, before the projective geometry,
- XYZ, the point cloud in the ego-vehicle coordinate frame, after the projective geometry.

The first shows a more compact and dense point cloud in the background despite presenting several distinct vehicles. The basic principle on which DBSCAN operates

is to identify densely grouped points separated by more sparse areas. This explains the failure for DBSCAN in RDC features space to successfully cluster vehicles that are close to each other in the back.

On the contrary, in the XYZ features space, the scattering in the background decreases the connectivity between the points, despite them belonging to the same entity. This suggests the need for an increased value of ε to find the required minimum number of points to form a cluster in a larger area.

This means that the neighbourhood radius is here directly dependent on the projective geometry and the spatial resolution of the camera. Thus, the necessary values for ε could be derived theoretically from the parameters of the cameras and stereovision set-up. The spatial resolution gives a point-to-point distance, indirectly estimating the density of the area.

In other words, say a minimum of 10 points is required to form a cluster, this is equivalent to requiring a minimum of 10 pixels in the image. Knowing the cameras parameters, and given a distance, the width covered by these 10 pixels can be calculated and would correspond to a rough estimation of twice ε.

Considering one camera's projective geometry, shown in Chapter 2, as the light rays gather through the pinhole, the further away one looks, the larger the width captured on one pixel. This can explain the scattering that is continuously observed on projections from disparity maps. This spatial resolution, the horizontal width per pixel w_{pp}, is expressed mathematically depending on the baseline b of the stereovision set-up and the disparity d in Equation (5.10).

$$w_{pp} = \frac{b}{d} \tag{5.10}$$

Two limitations can be observed from Equation (5.10):

- when $d = 0$, which is infinity, the spatial resolution can not be computed. This is normal and w_{pp} tends towards infinity too.
- when $d = 1$, which are far away points, the estimation of the spatial resolution is limited by the stereovision set-up.

As illustrated in Figure 5.9, the width per pixel increases with distance. It changes the gap between the point in the vehicle point clouds, i.e. the density, and thus the search radius ε for neighbour points. For example, if the minimum number of points to form a cluster is 9, then ε could be estimated by $\frac{3}{2}w_{pp} = \frac{3}{2}\frac{b}{d}$.

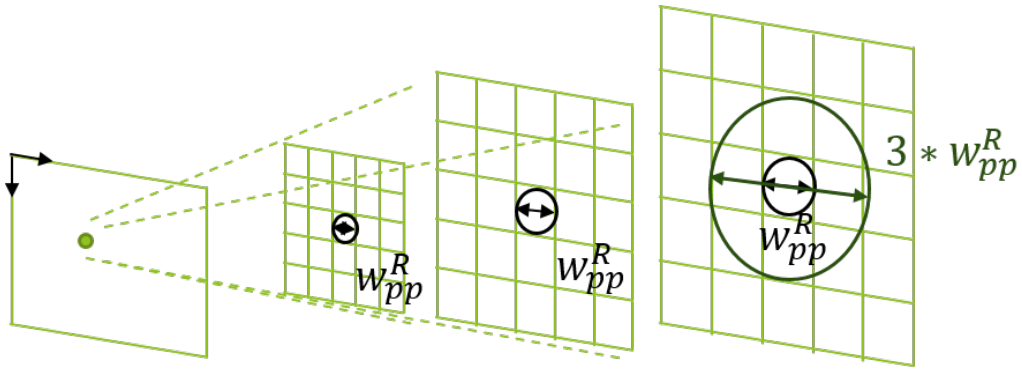


Figure 5.9: Visual interpretation of width per pixel, increasing with distance R.

In my experiments, $MinPts$ is set to 80, and in the Cityscapes, cameras were distanced by an "automotive-grade 22cm baseline" [65]. Following the above reasoning, simplifying the neighbourhood sphere to a one dimensional problem (horizontal width), I estimate ε at 0.29 m for a distance of 50 m.

Values of ε were saved when running DBSCAN on the XYZ features space in Section 5.4.1. Remember that in the target data, the distance is limited to 50 m. The majority of the global values ranged from 0.1 m to 0.7 m, with the mean and median at 0.31 m, the 25th percentile at 0.2 m and the 75th percentile at 0.4 m.

Values of ε were also saved in the next chapter when DBSCAN is run as part of the pipeline designed in Section 3.2. The selected values are displayed in Figure 5.10 depending on the number of points encountered in the point cloud. The order of magnitude is the same as on the target data, with values approaching 0.4 m and 0.5 m on average.

As they are global values in the current experiments, this numerical comparison only allows to conclude that this is the same order of magnitude, confirming the potential of using this link for parameter setting. However, directly using equation (5.10) to set-up the ε parameter for it to adapt to the density variation of the point cloud remains to be tested. Note that $MinPts$ should also adapt to the disparity to account for

perspective.

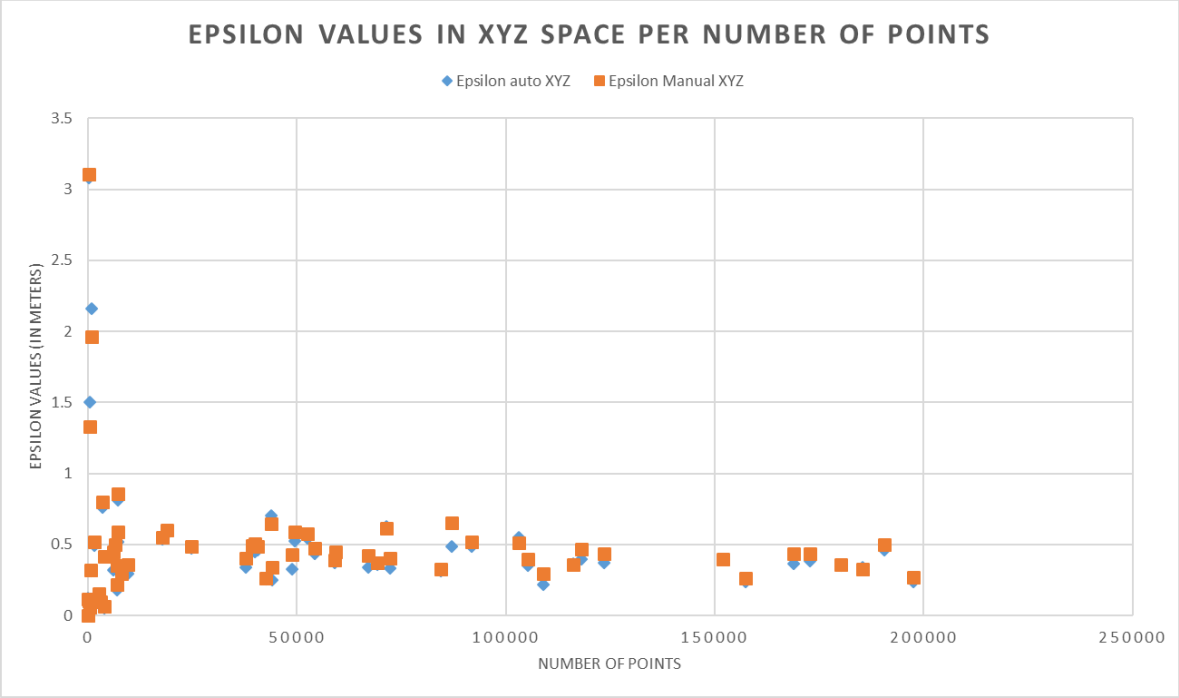


Figure 5.10: Values selected for ε in XYZ space.

It is important to observe couple of things here in the context of self-driving cars. The parameter ε has been linked to tangible hardware set-up. It is likely that a similar reasoning could be applied to LiDAR data by introducing the speed rotation and angle steps of the laser beam.

Both this assumption and what I presented need further investigation and extensive testing. However, to my knowledge, it is the first time a link of this nature is established. Indeed, in order to use DBSCAN, the selection of the parameters is usually automated and thus they are not set analytically [97] [100] [101].

This is important because self-driving cars need to be reliable, dependable, but also explainable. The technology has a goal to be safer than human drivers, but in case of a failure the technology needs to offer ways to trace and identify issues. Demystifying parameters selection is a step in that direction.

This association between software parameters and hardware parameters would not be that straightforward with the embeddings learned by neural networks in between like in most current techniques.

5.4.3 Post-processing

In previous Section 5.4.1, changing the feature space to XYZ allowed DBSCAN not to group all the background vehicles into a single cluster due to the higher density in RCD feature space. However, as a consequence, DBSCAN tends to identify more vehicles in the background, some correctly, but also increasing the number of false positives.

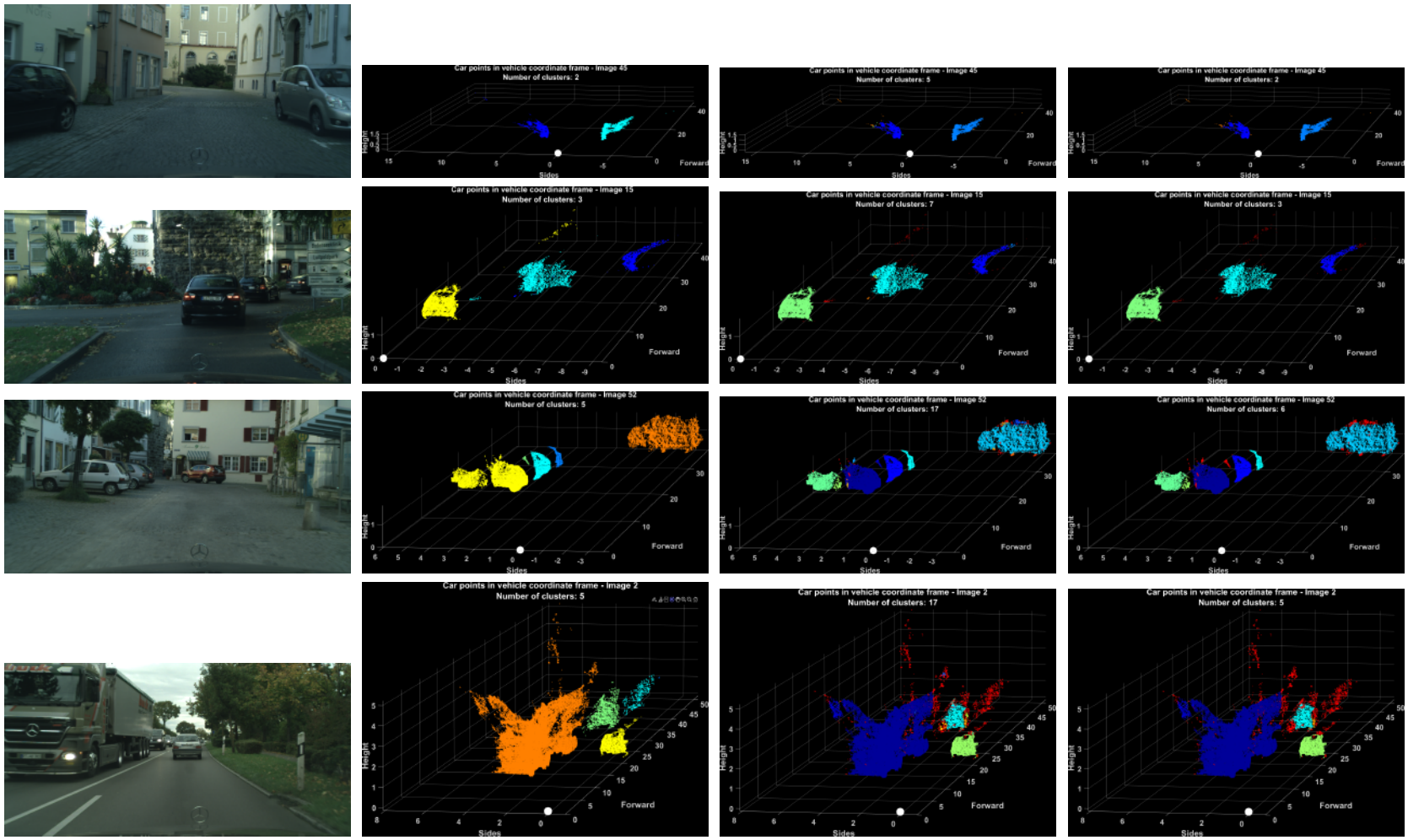
Indeed, when the disparity map is inaccurate, especially around the contour of each instance, the scattering after projection in 3D is generally more important. While DBSCAN recognises the noise, some small groups of points within the noise meet the $MinPts$ criteria, forming a cluster. Also, the radius ε becomes too small for the scattering in the background, breaking the vehicles into several clusters. As a result, DBSCAN tends to detect multiple "pockets of noise" as a vehicle instance. This effect is displayed in the 4^{th} column of Figure 5.11.

These detection are spatially incoherent because they are either too small to be a vehicle or correspond to a vehicle already detected and identified with another cluster. These "pockets of noise" should either be actual noise or part of the closest cluster. This is similar to multiple detection in neural networks, which are corrected using non-maximum suppression to keep the most probable detection.

Following an equivalent reasoning, filtering is implemented to discard these "pockets of noise", by applying a minimum volume threshold, empirically set to 0.1 m.

Hence, clusters identified by DBSCAN with a volume smaller than 10 cm are then considered as noise. Note that this is not equivalent to considering that a vehicle of size 10 cm is coherent. Indeed, this is accounting for occluded vehicles, which more often than not do not appear in their entirety. So, this is considering that a detection smaller than 10 cm is most probably a false alarm.

The resulting output of DBSCAN post filtering is displayed in the 5^{th} column of Figure 5.11. For row 1 and 2, the number of identified clusters decreases from 5 to 2, and from 7 to 3 respectively, removing all the false positives; only the expected and correct detection remain. In row 3 and 4, the number of clusters is reduced from 17 with 12 false detection each, to 6 clusters among which 5 are correct detection, and to 5 clusters with 3 correct detection, respectively.



From left to right: left image, target data, non filtered DSBCAN, filtered DBSCAN.

Figure 5.11: Filtering multiple outputs of DBSCAN.

Thus, this does not remove all the wrong multiple detection. However, it removes the majority of them while keeping the correct detection of small instances, like the furthest parked vehicle on the left in cyan in row 3.

5.5 Summary

In this chapter, I investigated the fourth step of the pipeline I developed: clustering. The objective was to identify each object entities within a point cloud of a given class of interest - i.e. *vehicle*.

The experiments in this chapter were made on the target data in order to remove the influence of the semantic segmentation step and assume an ideal point cloud formation. Two techniques have been explored: GMM and DBSCAN.

Although GMM has the benefit of speed and adapts to the features distribution, it is limited by its inability to generalise on all possible scenarios encountered in autonomous driving application. Mainly, it does not find the appropriate number of cluster on its own and the co-variance matrix does not fully encompass the objects shapes. Indeed, GMM will inherently divide single entities into several clusters without any theoretical mean to mitigate this.

Instead, DBSCAN demonstrates potential by not only defining clusters independently on raw data but also by identifying noise even within the target data. Unlike GMM, occasional division of single entities can be addressed through parameters selection and noise post-processing.

DBSCAN is challenged by the density variations in disparity values as it processes points from foreground to background. The feature space in which the clustering is performed plays a role in the accuracy of the instance segmentation.

Further work includes leveraging the pixel intensity information as well as linking DBSCAN parameters to the camera projection.

6. CASE STUDY - FRAMEWORK VALIDATION

An innovative framework for 3D panoptic segmentation was developed and tested in Chapter 3 for *vehicles*, revealing some potential but lacking reliability.

In particular, the point cloud aspect did not always resemble the target data by including points that did not belong to the *vehicle* class. This inclusion was caused by an erroneous semantic segmentation. In Chapter 4, the architecture of the deep neural network was modified to enhance the segmentation: layers were added and the classification problem simplified to a specific *vehicle* detector. The boundary F-score especially increased as a result, which positively impacted the aspect of the resulting point cloud.

In addition, the preliminary test demonstrated that the initial clustering method used, GMM, was challenged to identify instances correctly. In Chapter 5, further investigations established that GMM suffered from limitations inherent to its nature and that it is not the most appropriate method for this application. On the contrary, the study conducted in Chapter 5 determined that DBSCAN was more suitable and adaptable for autonomous driving.

In this chapter, the proposed framework is again tested with the newly integrated improvements developed in Chapter 4 and Chapter 5 for each of the corresponding step.

6.1 Experiment

The case study is carried out on the same data as the preliminary tests in Chapter 3: the city Lindau from the Cityscapes dataset, using the target data generated in Section 3.3.4 for comparison purposes.

Step 1. For semantic segmentation, the deep neural network SegNet [41] is used, with a deeper encoder: VGG19 [40]. The network is fine-tuned on the Cityscapes dataset [65] to recognise and segment *vehicle* pixels in particular.

Step 2. This step is unchanged and the disparity maps are still pre-computed and provided by the Cityscapes dataset. The method applied is Semi-Global Matching

(SGM) [69].

Step 3. This step is unchanged and the point cloud is created with the *vehicle* class, keeping only point within a 50 m distance.

Step 4. For instance clustering, DBSCAN is used and is run with the data in the X, Y, Z feature space. The neighbourhood radius ε is automatically selected for each scenario, and the post-processing described in Section 5.4.3 is implemented to remove spatially incoherent multiple detection.

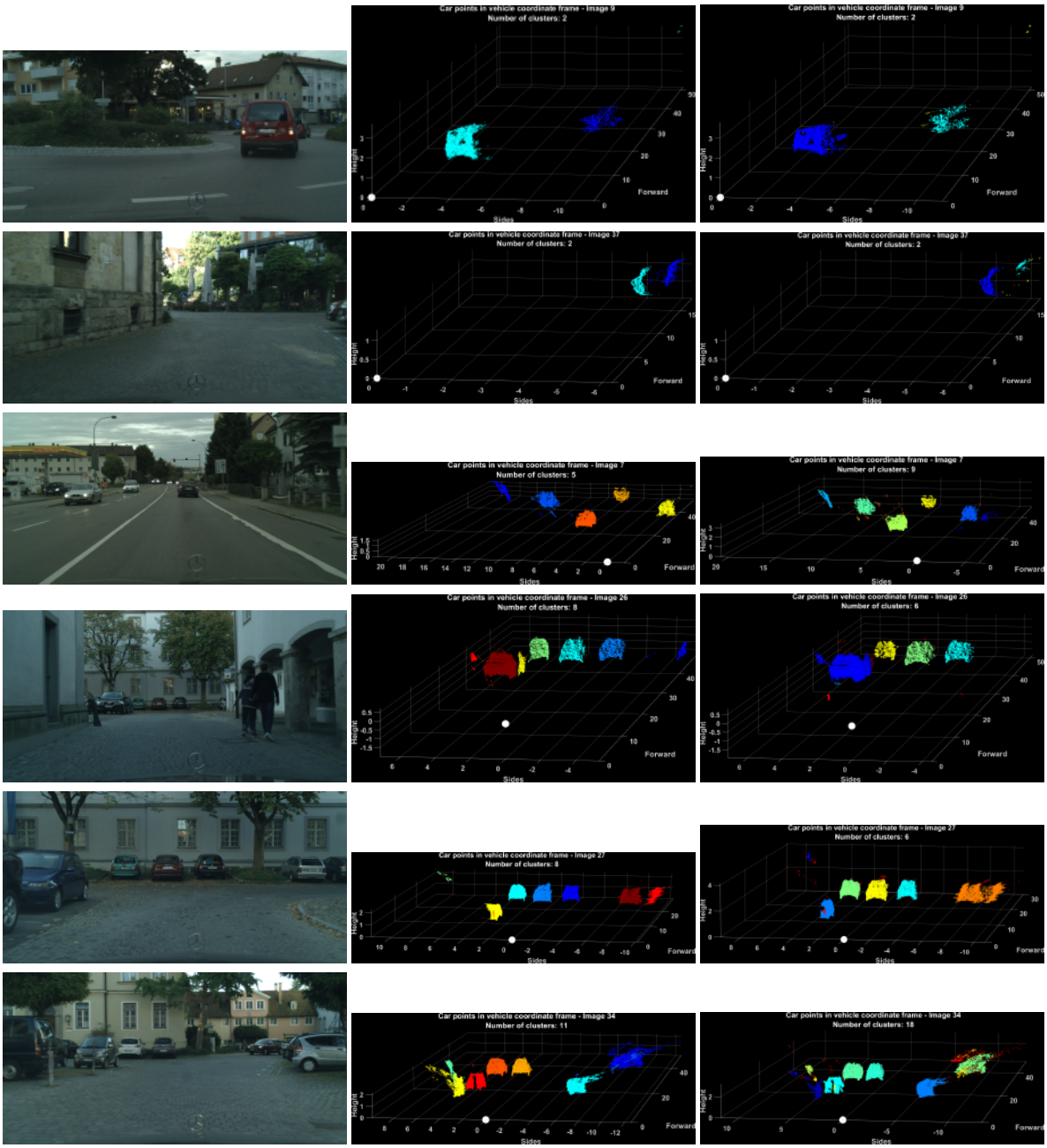
To summarise, the composition of the pipeline for this preliminary test is the following:

1. Semantic Segmentation: SegNet, VGG19 encoder, binary classification
2. Disparity map: Semi-Global Matching
3. Filtering: vehicle under 50 m
4. Clustering: DSBCAN on XYZ space with post-processing
5. Mapping: based on the Cityscapes dataset

6.2 Results

I present in this section the results of the case study conducted to test the pipeline created in Section 3.2 with the improvements discovered in Chapter 4 and Chapter 5. Like in Section 3.4.2, I display in Figure 6.1 and Figure 6.2 scenarios that condense in a few examples recurrent behaviours and patterns observed during result analysis. Both point cloud aspect and instances division are again inspected.

The aspect of the point clouds are greatly enhanced from the preliminary tests previously introduced in Section 3.4.2. The majority of the scenarios resemble the target data, sometimes perfectly like in Figure 6.1 row 1 and 2 despite the relative difficulty of the task, specifically in row 2. When falsely detected points are included in the *vehicle* point cloud, they regularly correspond to added points around the contours of the vehicles such as in Figure 6.1 row 3, 5 and 6, as well as Figure 6.1 row 3 and 5. In these cases, DBSCAN is capable of recognising these points as noise, mitigating small errors in the semantic segmentation.

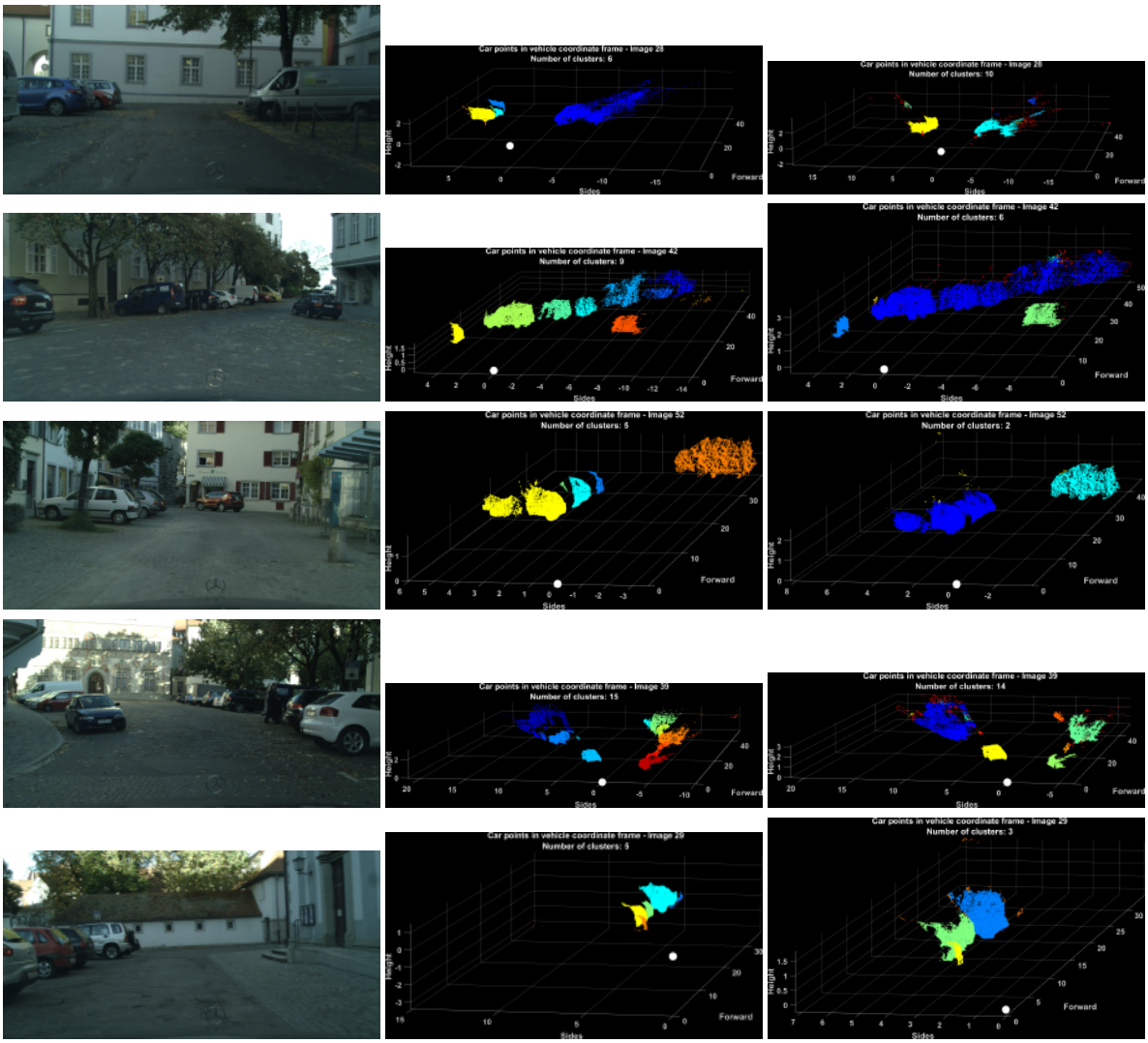


From left to right: left image, target data, pipeline output with: SegNet19 vehicle detector, filtered DBSCAN on XYZ.

Figure 6.1: Case study 2 1/2

On occasions, in Figure 6.1 row 3, 4 and Figure 6.2 row 1, small groups of points are falsely detected but not considered as noise, resulting in a false detection.

In Figure 6.1 row 1, some points in cyan in the target data, coming from the red



From left to right: left image, target data, pipeline output with: SegNet19 vehicle detector, filtered DBSCAN on XYZ.

Figure 6.2: Case study 2 2/2

van, are wrongly placed because of errors in the disparity maps. These errors are caused by the van's back window which is a transparent surface. Thus, the detected features in the disparity maps correspond to the building seen through the window. These errors cannot be corrected as not "wrong" per se. However, DBSCAN correctly identifies these points as noise, producing a better object map than the target data. This also results in a more coherently sized-vehicle. This was not possible with GMM.

In Figure 6.1, row 3, 4, 5 and 6, when the vehicles are visibly well separated, satisfying the density variations' principle on which DBSCAN operates, all the vehicles are well divided and identified. They also correspond to the high priority vehicles defined in Section 3.1: the vehicle immediately in front in the ego-vehicle's driving lane, the vehicles in the opposite lane and the parked vehicles whose status can change from *static* to *dynamic* and enter the ego-vehicle's lane.

These results for 3D panoptic segmentation are promising for navigation purposes. However, despite the changes made in Section 5.4 by introducing the XYZ feature space and adding post-processing filtering, the pipeline suffers couple of drawbacks in specific situations that are demonstrated in Figure 6.2.

First, in Figure 6.2 row 1 and 4, the aspect of the point cloud is affected by a significant amount of scattering. In these cases, the target data also show these errors so the semantic segmentation is correct but the disparity maps are not. The scattering results from the reflective surfaces from vans, which do not have enough distinctive features to produce an accurate disparity estimation.

It is worth noting though that despite the scattering, the majority of the vehicle - in cyan in row 1, in dark blue in row 4 - is detected, so this type of error inherent to visual data does not yield a missed detection. In term of safety, this is an essential characteristics. However, it results in multiple detection with "pocket of noise" that are too large to be filtered out by post-processing.

Second, in all scenarios of Figure 6.2, parked vehicles on the sides that appear occluded in images are never separated in individual entities. Although, the points extracted from the semantic segmentation are correct, DBSCAN fails to identify areas of low density between them, consider them all connected because they are too close to each other - less than ε. As a result, all the parked vehicles are grouped as one single vehicle.

This was not observed in previous experiments - not in the preliminary test with GMM, nor on the target data with DBSCAN. This can be explained by two reasons:

- GMM is given the number of clusters to be found,
- the target data was filtered to be an ideal point cloud.

Asking GMM to cluster into a specific number of clusters forces it to divide these connected vehicles into more vehicles than it might naturally do if the number of clusters was adjusted with information criterion measures. It makes it here more efficient than DBSCAN but not necessarily because it understands better the data structure.

Remember from Section 3.3.4 that the target data was filtered in order to obtain an ideal point cloud for comparison purposes. The filtering included a morphological operation on the contours of each instance, dependent on both distance and size of vehicles. As a consequence, incorrect disparity values were partially removed but this also enhanced the low density areas between instances. This facilitated the task for DBSCAN to identify instances on the target data in Chapter 5.

As this filtering was implemented in order to mitigate potential inaccuracies from the disparity maps, further work includes changing the SGM method used for disparity estimation to a more accurate and recent technique. For both observed issues, scattering and connected vehicles, depth estimation in the developed framework needs further investigation.

6.3 General considerations

In this section, both limitations and advantages of the developed framework are discussed in a more general discussion.

6.3.1 Limits of the framework

One hypothesis of this work was that reasoning in 3D would help divide close vehicles that appear overlapping or occluded in images because vehicles are naturally separated in 3D unless a crash occurs. The results in this case study show that this is not as straightforward. However, this hypothesis is not invalidated yet for several reasons.

First, NVIDIA [4] [3] explores a similar hypothesis that showed promising results. These approaches clearly differ from this thesis by the nature of the data and techniques used. Indeed, their bird's eye views are Lidar point clouds and their pixel-wise panoptic segmentation on images is entirely based on deep neural networks. This

thesis however uses solely visual data to create point clouds and is based on unsupervised clustering to perform 3D instance segmentation.

Second, as shown in Section 3.3.4 on the target data generation, errors were still visible on some example due to incorrect disparity values. The second step of the pipeline - estimating disparity maps - requires further investigation in order to determine its influence over the quality of the 3D panoptic segmentation.

Third, only geometrical information was utilised as input features for the clustering algorithms. These features included either rows, columns and disparity values, or the point coordinates X, Y, Z, depending on the feature space. In further work, adding other features such as pixels colour intensity would leverage the usage of cameras over a simple laser scan. Interestingly, multi-spectral imaging could also be explored in order to not only differentiate between colours but, potentially, also between car paints if two overlapping vehicles happen to be of a similar colour.

To summarise, if only geometrical features are used, projecting in 3D to reason in that space using visual data may not be sufficient in order to separate overlapping instances when objects are distant. It is likely to require other conditions in the framework in order to successfully divide and identify distant overlapping instances. However, this point should be pondered as distant objects are not critical to navigation.

Finally, the developed framework using only cameras data is limited by stereovision. The precision of the estimations in depth, location and object size are limited by the accuracy of the disparity estimation, which only decreases with distance. This therefore limits the distance range for scene perception.

6.3.2 Advantages of the framework

Unlike the majority of the methods currently used, the developed pipeline does not require the production of embeddings prior to cluster the points into separate instances. As a result, this pipeline does not need as much training, i.e. not as much data and time too. Only the semantic segmentation step needs training.

However, object recognition is a task for which the accuracy of available convolutional networks have surpassed the Human capabilities already. The transfer learning from

pre-trained network on large scale databases makes the preparation of those networks faster. As a result, this pipeline has the potential to be easily implemented and trained.

Similarly, the main characteristic of the pipeline is to be modular. In this work, I have shown its possibilities with state of the art techniques. However, considering the pace of progress exhibited by the research community, this modular pipeline can leverage existing benchmarks for each of the framework's steps. This includes better accuracy in both depth estimation and contour definition for semantic segmentation, which would allow a more realistic point cloud to work with during clustering.

This framework provides directly in 3D the profile of each instance for panoptic segmentation which may help the generation of environment representation, whether detailed maps or occupancy grids. This should help in the more general software stack of the autonomous car in order to perform the driving task itself, in particular navigation and obstacle avoidance during path planning.

Furthermore, DBSCAN can be applied directly on 3D point clouds and is already applied on LiDAR point clouds. I created a classified point cloud based on visual data allowing DBSCAN to be also applied with cameras. Finding processing algorithms that has a potential for both hardware set-up means it could be easier to reach a technology consensus.

Also, as this classified point cloud created by my approach is sparse, it can be processed at once. This is a significant edge in term of computational time compared to lidar point clouds, which often require to be processed by 1 m square portions [56].

That being said, CNN applied to images usually offer fast inference execution but at the cost of transparency and lack of interpretability. In this work, ε is linked to the stereovision setup for the first time. So on the contrary, linking the clustering reasoning to a tangible hardware parameter here helps demystifying the parameter selection in the processing.

7. CONCLUSION

This thesis presented a novel framework for 3D panoptic segmentation based on visual data. The framework leverages unsupervised clustering methods as well as disparity information with an innovative perspective addressing some of the gaps of similar approaches in the literature. The proposed framework is designed as a 5 steps pipeline that performs instance clustering on a 3D classified point cloud created from 2D image analysis. The image processing is comprised of a semantic segmentation task used to extract relevant points from a disparity map. The instances' shapes are then mapped with respect to the ego-vehicle.

This innovative framework was tested with state of the art techniques for each of the modular step of the pipeline. These tests were conducted on a dataset of urban traffic scenes appropriate for autonomous driving application. For qualitative comparison, the framework was tested on one type of dynamic objects - *vehicles* - and the generation of a "target" point cloud was introduced.

The framework in its initial version demonstrated that it can successfully classifies, detects and identifies vehicle instances in simple and ideal scenarios. However, it also suffered some drawbacks, notably in the vehicle point cloud formation and in the instance clustering. Different steps in the pipeline have been explored further to improve the framework's behaviour on 3D panoptic segmentation. In particular, the semantic segmentation and the instance clustering.

In semantic segmentation, the objective was to classify each pixel in the image, for which a convolutional neural network was used, SegNet. The depth of the network was investigated by adding layers, and the complexity of the task by initially performing multiple objects classification and then binary classification - as a vehicle detector specifically.

My experiments demonstrated that when training with limited resources, favouring deeper networks on a simpler task resulted in a more accurate semantic segmentation with a significant increase in the precision of the boundary F-score (+23 pp on average). In the context of the developed pipeline, this enhancement in contour definition effected in a vehicle point cloud containing less noise and thus its aspect

resembles more that of the target data.

As for instance clustering, the objective was to identify each object entities within the generated *vehicle* point cloud merging both the semantic segmentation and the disparity maps. In order to compare different unsupervised clustering techniques without the influence of the previous steps and assume an ideal point cloud formation, the experiments were made on the “target” data. Two techniques have been explored: GMM for distribution clustering and DBSCAN for spatial clustering.

Although GMM has the benefit of speed and adapts to the features distribution, it is limited by its inability to generalise on all possible scenarios encountered in autonomous driving application. Mainly, it does not find the appropriate number of cluster on its own and the co-variance matrix does not fully encompass the objects shapes.

Instead, DBSCAN demonstrates promising outcomes by not only defining clusters independently on raw data but also by identifying noise even within the “target” data. Unlike GMM, occasional division of single entities can be addressed through parameters selection and noise post-processing.

However, DBSCAN may be challenged by distant objects, although they are less important for navigation, because of the inherent decrease in disparity accuracy resulting from the vanishing point problem. Furthermore, DBSCAN is also challenged by overlapping vehicles that are close to each other and appear connected because this type of scenario does not allow for a clear area of low density between them. Although, the feature space in which the clustering is performed plays a role in the accuracy of the instance segmentation, these imperfections in the point clouds are inherent to the usage of visual data.

7.1 Contributions

This work tried to smartly make use and combine state of the art technique to provide a robust and transparent process for situational awareness of autonomous vehicles. In particular, for navigation purposes, visual perception is applied to find **where** and **what** are the surrounding dynamic objects. The contributions made through this framework’s development are detailed below.

First and foremost, to my knowledge, it is the first time that disparity maps are combined with pixel-wise segmentation into a 3D classified point cloud that is used for processing and not just mere visualisation, unlike for laser data. This work demonstrated that disparity maps could be used in a different way, not passed as an input of the neural network, nor used as simple “look-up tables” to retrieve depth information after detection.

It is also the first time that unsupervised clustering is applied on this “raw” data from visual data to obtain objects instances and that panotpic segmentation is not entirely performed based on deep neural networks.

My work shows that the embeddings learned through deep neural networks are not a necessity. Without these features representation, my work establishes for the time a link between DBSCAN parameters and the cameras parameters.

Additionally, this thesis showed the ability to provide a well distinguishable 3D profile of the detected object which is a tremendous improvement compared to 3D rectangular and uniform bounding boxes being currently the best achieved representation to my knowledge. My work highlighted the importance of the BF score metric in a domain where it was not thought as crucial, demonstrating the viability of segmentation in autonomous driving. This opens new perspectives in semantic segmentation to improve architectures, or loss function designs for better and more efficient training of the networks.

Finally, I developed a modular pipeline that leverages existing benchmarks that were created to facilitate progress in the domain. The different aspects, limitations and advantages of the developed framework were discussed to allow the community to use it. The code will be made available as a MATLAB toolbox.

7.2 Future work

As the framework was developed for 3D panoptic segmentation while the task was not formally defined yet, there are several possible course of actions for further work, as well as improving each of the steps of the pipeline. A non exhaustive list is outlined in this section.

A limitation in using predefined benchmarks is that sometimes the evaluation met-

rics keep being used to fit the benchmarks and provide assessment comparison rather than because they are the best fit for the task and application at hand. While this framework has been assessed qualitatively and its behaviour explained, there is need for it to be properly (i.e. quantitatively) evaluated. To do so, the appropriate datasets with the corresponding 3D labels and a metrics appropriate for 3D panoptic segmentation need to be either found or defined.

Regarding the framework itself, possible improvements include:

- Depth estimation (step 1): it was seen that disparity maps approximation can affect the quality of the classified point cloud. Instead of using SGM to generate disparity map for stereovision based depth estimation, alternative and more performing approach could be integrated. The change to a more performing and recent technique [103] can be based on the corresponding benchmark.
- Semantic segmentation (step 2): if SegNet was chosen and compared with two encoder versions (VGG16, VGG19), there is nothing preventing the integration of an alternative trained neural network on urban scene segmentation in the proposed framework. As for step 1, this change can be based on the performance observed on the corresponding benchmark.
- Scene filtering (step 3): the pipeline can be implemented for other classes than *vehicles*, such as *pedestrians* or *bicycles*.
- Clustering (step 4):
  - On GMM approach: there are several improvement that may be implemented and that I have not been able to try because of time constraint:
    - apply zero-mean normalisation on the features,
    - change the initial conditions,
    - use information criterion measures,
    - pre-estimate the clusters.
  - On DBSCAN: Despite demonstrating the adequateness of this approach on urban traffic vehicle distribution in space, there are still possible improvements comprising:

* adapt the algorithm to handle density variations within the point cloud
* change the parameter of the neighbourhood radius ε,
* integrate the use of additional features such as pixel's colour information.

It is interesting to note that, although GMM can be further improved, its very nature may prevent it to match an improved DBSCAN due to its limitations highlighted in Section 5.3. Also note that changing the parameter ε could be done in different manners as well: using the width per pixel described in Section 5.4.2, or using a multi-dimensional search radius.

Finally, the above list of improvements demonstrate the modularity of the proposed framework that do not rely on a single approach, but can evolve and interchange techniques for the different step of the pipeline accordingly to advancement on the related specific component of the pipeline.

REFERENCES

[1] Chris Urmson et al. “Autonomous Driving in Traffic: Boss and the Urban Challenge”. In: *AI Magazine* 30.2 (2009), p. 17.

[2] Lawrence Burns. *Autonomy, the quest to build the Driverless car - and how it will reshape our world*. London: William Collins, 2018.

[3] Neda Cvijetic. *Laser Focused: How Multi-View LidarNet Presents Rich Perspective for Self-Driving Cars*. 2020. URL: https://blogs.nvidia.com/blog/2020/03/11/drive-labs-multi-view-lidarnet-self-driving-cars/.

[4] Neda Cvijetic. *Pixel-Perfect Perception: How AI Helps Autonomous Vehicles See Outside the Box*. 2019. URL: https://blogs.nvidia.com/blog/2019/10/23/drive-labs-panoptic-segmentation/.

[5] Gabriela Csurka, Diane Larlus, and Florent Perronnin. “What is a good evaluation measure for semantic segmentation?” In: *Procedings of the British Machine Vision Conference 2013*. Bristol, UK: British Machine Vision Association, 2013, pp. 32.1–32.11.

[6] *The PROMETHEUS project launched in 1986: Pioneering autonomous driving*. URL: https://media.daimler.com/marsMediaSite/en/instance/ko/The-PROMETHEUS-project-launched-in-1986-Pioneering-autonomous-driving.xhtml?oid=13744534.

[7] Sebastian Thrun et al. “Stanley: The Robot That Won the DARPA Grand Challenge”. In: *The 2005 DARPA Grand Challenge: The Great Robot Race*. Ed. by Martin Buehler, Karl Iagnemma, and Sanjiv Singh. Springer Berlin Heidelberg, 2007, pp. 1–43.

[8] *AutoX*. URL: https://www.autox.ai/en/index.html.

[9] *Apollo*. URL: https://apollo.auto/.

[10] *Honda SENSING Elite safety system with Level 3 automated driving features*. URL: https://hondanews.eu/eu/et/corporate/media/pressreleases/329456/honda-launches-next-generation-honda-sensing-elite-safety-system-with-level-3-automated-driving-feat.

[11] *Oxbotica*. URL: https://www.oxbotica.com/.

[12] *Wayve*. URL: https://wayve.ai/.

[13] *Yandex Self-driving Cars*. URL: https://sdc.yandex.com/.

[14] *Tesla*. URL: https://www.tesla.com/en_gb.

[15] *Self-Driving Cars Specialisation by Coursera and University of Toronto*. URL: https://www.coursera.org/specializations/self-driving-cars.

[16] Alberto Elfes. "Using Occupancy Grids for Mobile Robot Perception and Navigation". In: *Computer* 22.6 (1989), pp. 46–57.

[17] Matthias Schreier. "Environment representations for automated on-road vehicles". In: *at - Automatisierungstechnik* 66.2 (2018), pp. 107–118.

[18] P Bender, J Ziegler, and C Stiller. "Lanelets: Efficient map representation for autonomous driving". In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. 2014, pp. 420–425.

[19] Alex Kendall et al. "Learning to Drive in a Day". In: *2019 International Conference on Robotics and Automation (ICRA)* (2019).

[20] Sorin Grigorescu et al. "A survey of deep learning techniques for autonomous driving". In: *Journal of Field Robotics* 37.3 (2020), pp. 362–386.

[21] K J Bussemaker. "Sensing requirements for an automated vehicle for highway and rural environments". BMD. TU Delft, 2014. URL: http://resolver.tudelft.nl/uuid:2ae44ea2-e5e9-455c-8481-8284f8494e4e.

[22] Lex Fridman. *MIT 6.S094: Deep Learning for Self-Driving Cars*. 2018. URL: https://deeplearning.mit.edu/.

[23] Dang Ha The Hien. *The Modern History of Object Recognition — Infographic*. 2017. URL: https://medium.com/@nikasa1889/the-modern-history-of-object-recognition-infographic-aea18517c318.

[24] Michael Warren, David McKinnon, and Ben Upcroft. "Online calibration of stereo rigs for long-term autonomy". In: *Proceedings - IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 3692–3698.

[25] Akhil Gurram et al. "Monocular Depth Estimation by Learning from Heterogeneous Datasets". In: *arXiv : 1803.08018* (2018).

[26] Adrian Kaehler and Gary Bradski. *Learning OpenCV 3, Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, 2016, p. 1024.

[27] Michael Calonder et al. "BRIEF: Binary Robust Independent Elementary Features". In: *Computer Vision – ECCV 2010*. Ed. by Kostas Daniilidis, Petros Maragos, and Nikos Paragios. Springer Berlin Heidelberg, 2010, pp. 778–792.

[28] D G Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, 1150–1157 vol.2.

[29] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features". In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz. Springer Berlin Heidelberg, 2006, pp. 404–417.

[30] E Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F Pereira et al. Vol. 25. Curran Associates, Inc., 2012.

[32] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[33] Matthew D Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks". In: *Computer Vision - ECCV 2014. Lecture Notes in Computer Science, vol 8689.* Springer, Cham, 2014, pp. 818–833.

[34] Fei-Fei Li and Andrej Karpathy. *CS231n: Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.stanford.edu/2019/.

[35] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 779–788.

[36] Joseph Redmon and Ali Farhadi. “YOLO9000: Better, Faster, Stronger”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 6517–6525.

[37] Christian Szegedy et al. “Going deeper with convolutions”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015, pp. 1–9.

[38] Ross Girshick et al. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2014, pp. 580–587.

[39] Kaiming He et al. “Mask R-CNN”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 2980–2988.

[40] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks For Large-Scale Image Recognition”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. San Diego, USA, 2015.

[41] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. “SegNet: A Deep Convolutional Encoder - Decoder Architecture for Image Segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.12 (2017), pp. 2481–2495.

[42] Liang-Chieh Chen et al. *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*. 2017.

[43] Pierre Sermanet et al. *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks*. 2013.

[44] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[45] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[46] Rohit Mohan and Abhinav Valada. *EfficientPS: Efficient Panoptic Segmentation*. 2020.

[47] Andres Milioto, Leonard Mandtler, and Cyrill Stachniss. “Fast Instance and Semantic Segmentation Exploiting Local Connectivity, Metric Learning, and One-Shot Detection for Robotics”. In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 5481–5487.

[48] Daniel Bolya et al. “YOLACT: Real-time Instance Segmentation”. In: *The IEEE International Conference on Computer Vision (ICCV)*. 2019, pp. 9157–9166.

[49] Jifeng Dai, Kaiming He, and Jian Sun. “Instance-Aware Semantic Segmentation via Multi-task Network Cascades”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 3150–3158.

[50] Zeeshan Hayder, Xuming He, and Mathieu Salzmann. “Boundary-Aware Instance Segmentation”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 587–595.

[51] Xiaodan Liang et al. “Proposal-Free Network for Instance-Level Object Segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.12 (2018), pp. 2978–2991.

[52] Bert De Brabandere, Davy Neven, and Luc Van Gool. “Semantic Instance Segmentation for Autonomous Driving”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2017, pp. 478–480.

[53] Alexander Kirillov et al. “Panoptic Feature Pyramid Networks”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 6392–6401.

[54] Jonas Uhrig et al. “Box2Pix: Single-Shot Instance Segmentation by Assigning Pixels to Object Boxes”. In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 292–299.

[55] Mengye Ren and Richard S. Zemel. “End-to-End Instance Segmentation with Recurrent Attention”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 293–301.

[56] Kosuke Arase, Yusuke Mukuta, and Tatsuya Harada. “Rethinking Task and Metrics of Instance Segmentation on 3D Point Clouds”. In: *IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. 2019, pp. 4105–4113.

[57] R Qi Charles et al. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 77–85.

[58] A Grenier et al. “Towards Scene Understanding Implementing the Stixel World”. In: *2018 IEEE British and Irish Conference on Optics and Photonics (BICOP)*. 2018, pp. 1–4.

[59] Timo Scharwächter et al. “Stixmantics: A medium-level model for real-time semantic scene understanding”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8693 LNCS.PART 5 (2014), pp. 533–548.

[60] Lukas Schneider et al. “Semantic Stixels: Depth is not enough”. In: *2016 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2016, pp. 110–117.

[61] Thomas M Hehn, Julian F P Kooij, and Dariu M Gavrila. “Instance Stixels: Segmenting and Grouping Stixels into Objects”. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019, pp. 2542–2549.

[62] Zygfryd Wieszok et al. “Stixel based scene understanding for autonomous vehicles”. In: *2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC)*. 2017, pp. 43–48.

[63] Hernán Badino, Uwe Franke, and David Pfeiffer. “The Stixel World - A Compact Medium Level Representation of the 3D-World”. In: *Pattern Recognition. DAGM 2009. Lecture Notes in Computer Science, vol 5748*. Ed. by J Denzler, G Notni, and H Süße. Springer, Berlin, Heidelberg, 2009, pp. 51–60.

[64] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for autonomous driving? The KITTI vision benchmark suite”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. Providence, RI, USA: IEEE, 2012, pp. 3354–3361.

[65] Marius Cordts et al. “The Cityscapes Dataset for Semantic Urban Scene Understanding”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 3213–3223.

[66] Daniel Scharstein and Richard Szeliski. *The Middlebury Computer Vision*. URL: `https://vision.middlebury.edu/`.

[67] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *Computer Vision - ECCV 2014. ECCV 2014. Lecture Notes in Computer Science, vol 8693.* Ed. by D Fleet et al. Springer, Cham, 2014, pp. 740–755.

[68] Alexander Kirillov et al. “Panoptic Segmentation”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 9396–9405.

[69] Heiko Hirschmüller. “Stereo Processing by Semiglobal Matching and Mutual Information”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.2 (2008), pp. 328–341.

[70] Stefan K Gehrig, Reto Stalder, and Nicolai Schneider. “A Flexible High-Resolution Real-Time Low-Power Stereo Vision Engine”. In: *Computer Vision Systems*. Springer International Publishing, 2015, pp. 69–79.

[71] Bryan C Russell et al. “LabelMe: A Database and Web-Based Tool for Image Annotation”. In: *International Journal of Computer Vision* 77.1-3 (2008), pp. 157–173.

[72] Mark Everingham et al. “The PASCAL Visual Object Classes (VOC) Challenge”. In: *International Journal of Computer Vision* 88.2 (2010), pp. 303–338.

[73] Mark Everingham et al. “The PASCAL Visual Object Classes Challenge: A Retrospective”. In: *International Journal of Computer Vision* 111.1 (2015), pp. 98–136.

[74] Gabriel J Brostow, Julien Fauqueur, and Roberto Cipolla. “Semantic object classes in video: A high-definition ground truth database”. In: *Pattern Recognition Letters* 30.2 (2009), pp. 88–97.

[75] Andreas Geiger et al. “Vision meets robotics: The KITTI dataset”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237.

[76] Timo Scharwächter et al. “Efficient Multi-cue Scene Segmentation”. In: *Pattern Recognition*. Springer Berlin Heidelberg, 2013, pp. 435–445.

[77] *MATLAB Automated Driving Toolbox*. URL: https://uk.mathworks.com/products/automated-driving.html.

[78] German Ros et al. “The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 3234–3243. URL: http://synthia-dataset.net/.

[79] Xuan Li et al. “The ParallelEye Dataset: A Large Collection of Virtual Images for Traffic Vision Research”. In: *IEEE Transactions on Intelligent Transportation Systems* 20.6 (2019), pp. 2072–2084.

[80] Hassan Alhaija et al. “Augmented Reality Meets Computer Vision: Efficient Data Generation for Urban Driving Scenes”. In: *International Journal of Computer Vision (IJCV)* (2018).

[81] Fisher Yu et al. “BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 2633–2642.

[82] Nils Gählert et al. *Cityscapes 3D: Dataset and Benchmark for 9 DoF Vehicle Detection*. 2020.

[83] J. Behley et al. “SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences”. In: *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*. 2019.

[84] Jens Behley et al. “Towards 3D LiDAR-based semantic scene understanding of 3D point cloud sequences: The SemanticKITTI Dataset”. In: *The International Journal of Robotics Research* 40.8-9 (2021), pp. 959–967.

[85] Pei Sun et al. “Scalability in Perception for Autonomous Driving: Waymo Open Dataset”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 2443–2451. URL: https://waymo.com/open/about/.

[86] R. Kesten et al. *Level 5 Perception Dataset 2020*. 2019. URL: https://level-5.global/level5/data/.

[87] Jakob Geyer et al. “A2D2: Audi Autonomous Driving Dataset”. In: *arXiv: 2004.06320* (2020). URL: https://www.a2d2.audi.

[88] Qianqian Fang. *JSONLab: a toolbox to encode / decode JSON files*. URL: https://www.mathworks.com/matlabcentral/fileexchange/33381-jsonlab-a-toolbox-to-encode-decode-json-files.

[89] Evan Shelhamer, Jonathan Long, and Trevor Darrell. “Fully Convolutional Networks for Semantic Segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (2017), pp. 640–651.

[90] Liang-Chieh Chen et al. “Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation”. In: *Computer Vision - ECCV 2018. Lecture Notes in Computer Science, vol 11211.* Springer, Cham, 2018, pp. 833–851.

[91] Alberto Garcia-Garcia et al. “A Review on Deep Learning Techniques Applied to Semantic Segmentation”. In: *arXiv: 1704.06857* (2017), pp. 1 –23.

[92] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *arXiv: 1502.03167* (2015).

[93] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning Internal Representations by Error Propagation”. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*. Ed. by David E Rumelhart and James L Mcclelland. Cambridge, MA: MIT Press, 1986, pp. 318–362.

[94] David Eigen and Rob Fergus. “Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-scale Convolutional Architecture”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Santiago, Chile: IEEE, 2015, pp. 2650–2658.

[95] Jian Zhang and Ioannis Mitliagkas. “YellowFin and the Art of Momentum Tuning”. In: *arXiv: 1706.03471* (2017).

[96] Amélie Grenier. *Visual Scene Understanding for Self-Driving Cars Using Deep Learning and Stereovision*. Cranfield Online Research Data (CORD). 2019.

[97] Chunxiao Wang et al. “An Improved DBSCAN Method for LiDAR Data Segmentation with Automatic Eps Estimation”. In: *Sensors* 19.1 (2019), p. 172.

[98] V. Spruyt. *A geometric interpretation of the covariance matrix*. URL: https://www.visiondummy.com/2014/04/geometric-interpretation-covariance-matrix/.

[99] Martin Ester et al. “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD’96. AAAI Press, 1996, 226–231.

[100] Amin Karami and Ronnie Johansson. “Choosing DBSCAN Parameters Automatically using Differential Evolution”. In: *International Journal of Computer Applications* 91.7 (2014), pp. 1–11.

[101] Manisha Naik Gaonkar and Kedar Sawant. “AutoEpsDBSCAN : DBSCAN with Eps Automatic for Large Dataset”. In: *International Journal of Advanced Computer Theory and Engineering (IJACTE)* 2.2 (2013).

[102] Mihael Ankerst et al. “OPTICS: ordering points to identify the clustering structure”. In: *Proceedings of the 1999 ACM SIGMOD international conference on Management of data - SIGMOD ’99*. New York, New York, USA: ACM Press, 1999, pp. 49–60.

[103] Andreas Geiger, Martin Roser, and Raquel Urtasun. “Efficient Large-Scale Stereo Matching”. In: *Lecture Notes in Computer Science*. Vol. 6492 LNCS. PART 1. 2011, pp. 25–38.