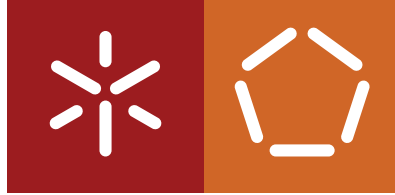


Universidade do Minho
Escola de Engenharia
Departamento de Informática

André Sousa Figueiredo

**LiDAR based 3D Object Tracking for
Autonomous Driving**

October 2022



Universidade do Minho
Escola de Engenharia
Departamento de Informática

André Sousa Figueiredo

LiDAR based 3D Object Tracking for Autonomous Driving

Master dissertation
Integrated Master's in Informatics Engineering

Dissertation supervised by
Paulo Novais (University of Minho)
Filipe Gonçalves (Bosch)

October 2022

COPYRIGHT AND TERMS OF USE FOR THIRD PARTY WORK

This dissertation reports on academic work that can be used by third parties as long as the internationally accepted standards and good practices are respected concerning copyright and related rights.

This work can thereafter be used under the terms established in the license below.

Readers needing authorization conditions not provided for in the indicated licensing should contact the author through the RepositóriUM of the University of Minho.

LICENSE GRANTED TO USERS OF THIS WORK:



CC BY

<https://creativecommons.org/licenses/by/4.0/>

ACKNOWLEDGEMENTS

This dissertation was only possible due to all the support I received from all the important people in my life. I want to thank my parents for their support and for understanding every time I was late or could not do my chores when they asked.

My brother and his girlfriend for their encouragement, my aunt for helping me on my everyday life and her kindness, and my girlfriend for her companionship and love, and for giving me the strength necessary during the difficult parts.

I would like to thank my supervisor, Professor Doutor Paulo Jorge Novais for this opportunity, for suggesting this topic and for his guidance.

I would to thank Bosch for the opportunity to work on the project *THEIA* and all my colleagues there, specially Luís Ferreira, for being my friend and for always being ready to discuss every matter.

Lastly, I want to thank Engenheiro Filipe Gonçalves, for his help and tutoring, which was fundamental to improve this dissertation.

André Figueiredo

This work is supported by European Structural and Investment Fund in the FEDER component through the Operational Competitiveness and Internalisation Programme (COMPETE 2020) [Project nº 047264; Funding Reference: POCI-01-0247-FEDER-047264].

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ABSTRACT

Technology has become essential for society's every-day-life and with the recent increase in artificial intelligence's interest, this area has gained more and more relevance for both people (e.g., due to the increasing number of users of personal assistants, such as *Siri*, *Alexa* and *Google Assistant*) and service providers (e.g., *Google* search engine and social networks' recommendation algorithms to keep users busy and active on their platforms - *Facebook*, *Youtube*, *TikTok*, etc.). Nevertheless, artificial intelligence has been applied to many other areas, such as targeted advertising to specific users, cybersecurity, medicine, and the automobile industry.

Although artificial intelligence has not been the perfect solution in the aforementioned applications, it has been responsible for several significant improvements in the last decade. For example, in the automobile industry, there are more and more companies offering solutions for autonomous vehicles, being *Tesla* the most notorious.

This evolution was driven by several factors, including need and interest in improving road safety, growing traffic problems that exist due to the increase of vehicles circulating, more reliable sensors, and recent advances in various areas of artificial intelligence, such as object detection, semantic segmentation, and object tracking.

These three areas are interconnected. However, they have different purposes - the first two (detection and segmentation) more related to static frame analysis (e.g., image based analysis), while object tracking is usually applied in dynamic environments (e.g., sequence of frames, such as a video) where its input is processed in order to track objects over time, allowing an intelligent system to be "aware" of its environment.

That said, this dissertation aims to study and explore the applicability and feasibility, as well as to develop and implement an object tracker in the context of autonomous driving. Furthermore, it is also intended to make a benchmark with state-of-the-art approaches and identify their main limitations.

The input data will be focused on Light Detection and Ranging (**LiDAR**) based 3D point cloud, as there are several datasets available, in particular KITTI [1], which, in addition to being widely used in the state-of-the-art, has also achieved positive results, even in real-time execution situations. However, these solutions usually require a lot of computational resources and, which can be a hurdle for its application in real-life settings.

KEYWORDS Object tracking, self-driving vehicles, LiDAR, deep learning

RESUMO

A tecnologia tornou-se essencial para o normal funcionamento da sociedade e com o recente aumento do interesse na inteligência artificial, esta área tem ganho cada vez mais relevância tanto para as pessoas (por exemplo, devido ao aumento do número de utilizadores de sistemas como assistentes pessoais, como *Siri*, *Alexa* e *Google Assistant*) como para os prestadores de serviços (por exemplo, o motor de busca da *Google*, os algoritmos de recomendação de várias redes sociais para manterem o utilizador ocupado e ativo nas plataformas em questão - *Facebook*, *Youtube*, *TikTok*, entre outros). Ainda assim, a inteligência artificial tem sido aplicada a muitas outras áreas, como, por exemplo, publicidade adaptada a cada utilizador, cibersegurança, medicina e indústria automóvel.

Embora a inteligência artificial não tenha sido a solução perfeita nas aplicações mencionadas acima, esta tem sido responsável por vários avanços significativos na última década. Por exemplo, na indústria automóvel, existem cada vez mais empresas que oferecem soluções para veículos autónomos, sendo a *Tesla* a mais reconhecida. Esta evolução foi alimentada por vários fatores, como a necessidade e interesse em melhorar a segurança na estrada, os crescentes problemas de trânsito que existem devido ao aumento de veículos a circular, sensores mais fidedignos e os avanços recentes em várias áreas da inteligência artificial, como, por exemplo, a deteção de objetos, segmentação semântica e *tracking* de objetos.

Estas três áreas estão interligadas. Contudo têm focos diferentes - as duas primeiras (deteção e segmentação) mais relacionadas com análise de frames estáticas (por exemplo, análise baseada em imagens), enquanto que o *tracking* de objetos é, usualmente, aplicado em ambiente dinâmicos (por exemplo, em sequência de frames, sendo vídeo um desses casos). Este input é então processado para executar a tarefa de monitorização, permitindo que um sistema inteligente esteja “ciente” do ambiente onde se encontra.

Posto isto, esta dissertação tem como objetivo estudar e explorar a aplicabilidade e viabilidade, assim como desenvolver e implementar um tracker de objetos no contexto da condução autónoma. Para além disso, também se pretende efetuar uma comparação com abordagens do estado de arte e identificar as suas principais limitações. Os dados de *input* serão focados em *Light Detection and Ranging* (LiDAR) baseado em *point cloud* 3D, uma vez que existem vários datasets disponíveis, em particular o KITTI [1], que, para além de ser muito utilizado no estado de arte, tem também alcançado resultados positivos, mesmo em situações de execução em tempo real. No entanto, estas soluções necessitam, normalmente, de muitos recursos computacionais, o que pode ser um entrave para a sua aplicação em contextos reais.

PALAVRAS-CHAVE *Tracking* de objetos 3D, condução autónoma, LiDAR, *deep learning*

CONTENTS

I INTRODUCTORY MATERIAL

1	INTRODUCTION	2
1.1	Motivation	2
1.2	Scope	2
1.2.1	Autonomous Driving	3
1.2.2	Perception on AI (LiDAR sensors)	4
1.2.3	Artificial Intelligence	5
1.2.4	Machine Learning	7
1.2.4.1	Deep Learning	7
1.2.4.2	Artificial Neural Networks	9
1.2.5	Object Detection	11
1.2.6	Tracking for Object Detection	13
1.3	Document structure	15
2	STATE-OF-THE-ART	16
2.1	Beyond Pixels	16
2.2	FANTrack	19
2.3	AB3DMOT	22
2.4	Probabilistic 3D Multi-Modal	25
2.5	PC-TCNN	27
2.6	Discussion and Analysis	29
3	OBJECTIVES	30
3.1	Goals	30
3.2	Work Plan	30
3.3	Contributions	32
II CORE OF THE DISSERTATION		
4	LIDAR BASED OBJECT TRACKING	34
4.1	Datasets	34
4.2	Dataset Analysis	36

4.3	Data Representation	43
4.4	Detection Model	44
4.5	Pre-Processing	46
4.6	Data Augmentation	49
4.7	Pipeline Implementation	51
4.7.1	Proposed Pipeline	52
4.7.2	First Tests	52
4.7.3	Implementation	53
4.7.3.1	Prediction	54
4.7.3.2	Association	56
4.7.3.3	Track Birth and Track Death	58
4.7.3.4	State Update	59
4.7.3.5	Non-Maximum Suppression	60
4.8	Evaluation Metrics	61
4.9	Hyper-parameter tuning	63
4.10	Overview	64
5	EXPERIMENTS AND RESULTS	66
5.1	Evaluation Process	66
5.2	Quantitative Analysis	67
5.3	Qualitative Evaluation	69
5.4	Resource Requirements	73
5.5	Ablation Study	73
6	CONCLUSIONS, LIMITATIONS AND FUTURE WORK	75
6.1	Limitations	76
6.2	Future Work	76
	Bibliography	78

LIST OF FIGURES

Figure 1	Proposed levels of AD according to [2].	3
Figure 2	Overview of the main branches of AI, extracted from [3].	6
Figure 3	Representation of multiple types of DL architectures.	9
Figure 4	Network and Neuron representations	10
Figure 5	Illustration of the gradient descent algorithm steps to minimise the cost function, extracted from [4].	11
Figure 6	Illustration of IoU, taken from [5].	13
Figure 7	Concept of 3D-2D and 3D-3D costs using 2 consecutive frames on the left and right respectively, extracted from [6].	17
Figure 8	2 Beyond Pixels tracking examples and respective results, extracted from [6].	18
Figure 9	FANTrack Similarity Network Pipeline, extracted from [7].	19
Figure 10	FANTrack proposed system pipeline, extracted from [7].	20
Figure 11	FANTrack tracking examples	22
Figure 12	ABD3DMOT proposed pipeline, extracted from [8].	23
Figure 13	ABD3DMOT example, extracted from [8]	25
Figure 14	Tracking example with wrong detections, extracted from [9].	27
Figure 15	Tracklet Pipeline, extracted from [10].	28
Figure 16	Example of KITTI dataset annotated 3D point cloud, obtained by [11]	36
Figure 17	Platform used for data acquisition, extracted from [12]	37
Figure 18	Number of objects per class	38
Figure 19	Classes distribution in percentage	39
Figure 20	Number of objects per class for both splits	39
Figure 21	Distribution of objects' lifespan	40
Figure 22	Cumulative distribution of objects' lifespan	41
Figure 23	Distribution of number of objects per frame	41
Figure 24	Cumulative distribution of number of objects per frame	42
Figure 25	Relative distance and positioning density	42
Figure 26	Objects dimensions (length, width and height) distribution	43
Figure 27	Illustration of most common techniques to encode 3D points, extracted from [13]	44
Figure 28	SE-SSD [14] architecture	45
Figure 29	PV-RCNN [15] architecture	46
Figure 30	Folder structure adaptations to comply with object detections' structure	47

Figure 31	Prediction information representation of a single object, orange represents ground-truth input data, while blue represents the expected output	48
Figure 32	Object association information representation, orange represents the original approach, while blue represents the final, both approaches return a number (confidence) ranging from 0 to 1	48
Figure 33	Examples of data augmentation techniques applied to images (a to f) and to bounding boxes (g to n)	51
Figure 34	Illustration of the proposed pipeline	52
Figure 35	Visual Representation of the expected output for a MOT, based on RGB and LiDAR annotated data (from GIF generated with [11])	53
Figure 36	Pipeline representation	54
Figure 37	Prediction model layers and number of parameters	55
Figure 38	Association model layers and number of parameters	57
Figure 39	Transition between different states	59
Figure 40	Object state updating with different weights, with blue, orange and black representing its detected, tracked and final state, respectively	60
Figure 41	Example of NMS's usage	60
Figure 42	Conversion error of 3D to 2D representation	67
Figure 43	Visualisation example, frame 63 of sequence 14 from KITTI Tracking dataset [1]	70
Figure 44	Example of a misclassified detection, labelling a parked bicycle as a Cyclist (ID 31), frame 81 of sequence 0 from KITTI tracking dataset [1]	70
Figure 45	Example of mislabelling a person walking with a bicycle as a Cyclist (ID 14), frame 111 of sequence 2 from KITTI tracking dataset [1]	71
Figure 46	Example of faulty detection where a road sign is predicted as a Car (ID 7), frame 27 of sequence 8 from KITTI tracking dataset [1]	72
Figure 47	Example of an occluded Car (ID 7) still being tracked, although it is no longer visible in the Light Detection and Ranging (LiDAR) data, frame 53 of sequence 1 from KITTI tracking dataset [1]	72

LIST OF TABLES

Table 1	Beyond Pixels results on the KITTI test set [1], extracted from [6].	18
Table 2	FANTrack results on the KITTI test set [1], extracted from [7].	20
Table 3	AB3DMOT results on the KITTI validation set [1] for “car” class, extracted from [8].	24
Table 4	AB3DMOT results on the KITTI validation set [1] for “pedestrian” and “cyclist” classes, extracted from [8].	24
Table 5	AB3DMOT results on the nuScenes validation set [16] over all classes, extracted from [8].	24
Table 6	Results on the nuScenes validation set [16] for multiple classes, extracted from [9].	26
Table 7	Results on the KITTI validation set [1] for “car” class, extracted from [9]. . . .	26
Table 8	Results on the nuScenes validation set [16] for “car” class, extracted from [9].	26
Table 9	PC-TCNN results on the KITTI benchmark, extracted from [10].	28
Table 10	PC-TCNN results on validation set with different number of point cloud frames, extracted from [10].	29
Table 11	Results of analysed architectures on KITTI benchmark [1], \uparrow denotes better.	29
Table 12	Results of analysed architectures on nuScenes benchmark [16], \uparrow denotes better.	29
Table 13	Work plan schedule.	31
Table 14	Dataset features comparison	35
Table 15	Comparison between implemented detection algorithms, AP values for medium difficulty, according to [17]	45
Table 16	Prediction component performance	56
Table 17	Association component performance	58
Table 18	2D and 3D HOTA results	68
Table 19	2D and 3D AP results (Moderate difficulty) for both detections’ as well as tracking’s bounding boxes	69
Table 20	Computational resources usage, regarding the tracking task	73
Table 21	Ablation experiments and respective Higher Order Tracking Accuracy (HOTA) performance	74

ACRONYMS

AD	Autonomous Driving
AI	Artificial Intelligence
AMOTA	Average Multiple Object Tracking Accuracy
ANN	Artificial Neural Network
AP	Average Precision
AV	Autonomous Vehicles
BEV	Bird's-Eye View
CNN	Convolutional Neural network
DL	Deep Learning
DLNN	Deep Learning Neural Network
FN	False Negative
FP	False Positive
FPS	Frames-per-second
HOTA	Higher Order Tracking Accuracy
IoU	Intersection over Union
IR	Infrared
LiDAR	Light Detection and Ranging
LSTM	Long Short Term Memory
mAP	mean Average Precision
ML	Machine Learning
MOT	Multi-Object Tracking
MOTA	Multiple Object Tracking Accuracy

NMS	Non-Maximum Suppression
OHOTA	Online Higher Order Tracking Accuracy
sAMOTA	Scaled Average Multiple Object Tracking Accuracy
SOT	Single Object Tracking
TL	Transfer Learning
TN	True Negative
TP	True Positive

Part I

INTRODUCTORY MATERIAL

INTRODUCTION

Every year, more than 1.35 million people die in road accidents worldwide [18], and between 20 to 50 million suffer non-fatal injuries, some resulting in disabilities. Road accidents are currently the leading death cause among people aged between 5 to 29 and the eighth for all age groups [18]. In addition, motorcyclists, cyclists, and pedestrians represent more than half of all road-related deaths. Usually, these accidents are caused by one of two factors: (i) human driver error (e.g., speeding, driving under the influence, distractions); and (ii) poor regulation (e.g., unsafe vehicles, inadequate post-crash care and poorly designed infrastructure).

1.1 MOTIVATION

One of the most talked-about and fascinating topics surrounding Artificial Intelligence (AI) is Autonomous Driving (AD) and self-driving vehicles, given that more and more people are adopting these solutions. One of the most relevant advantages in favour of AD is safety, since, according to statistics provided in [19], approximately 94% of road accidents in the United States are drivers-fault, so self-driving vehicles could decrease the number of deaths and accidents.

Even though we usually associate AD with a fully self-driving vehicle, technology has been used to make driving safer, e.g., automatic braking systems have been developed since the 1950s, however, it was first commercialised in 2003 by *Honda* [20]. In addition to this safety feature, many more are present in our vehicles, like rear cameras to help during parking, warnings for a close object, and even older features such as airbags and safety belts. Keeping that in mind, AD is the next step for the automobile industry as well as for transportation sector (an important industry sector in the economy that deals with the movement of people and goods).

1.2 SCOPE

In this section, the main concepts of this dissertation will be discussed, as well as the contexts where they are applied, and how they support in interpreting the current status of these thematics.

1.2.1 Autonomous Driving

When talking about Autonomous Vehicles (AV) and Autonomous Driving (AD), normally people associate these concepts with fully autonomous vehicles, however, that is not correct, since there are usually five to six AD levels [2], which are summarised in Figure 1:

0. *Manual driving*: driver is fully responsible for driving tasks of the vehicle;
1. *Driver assistance*: the vehicle is responsible for at least one driving aspect, e.g., cruise control or emergency brake assistance;
2. *Partial automation*: the vehicle is capable of performing several driving tasks, however, the driver must be permanently alert;
3. *Conditional automation*: the vehicle can perform every task in certain scenarios, but human monitoring is still required;
4. *High automation*: the vehicle is fully automated, performing all tasks, and the driver is dismissed, even though the driver can take control at any moment;
5. *Full automation*: no driver is needed, and the vehicle is autonomous in every scenario.

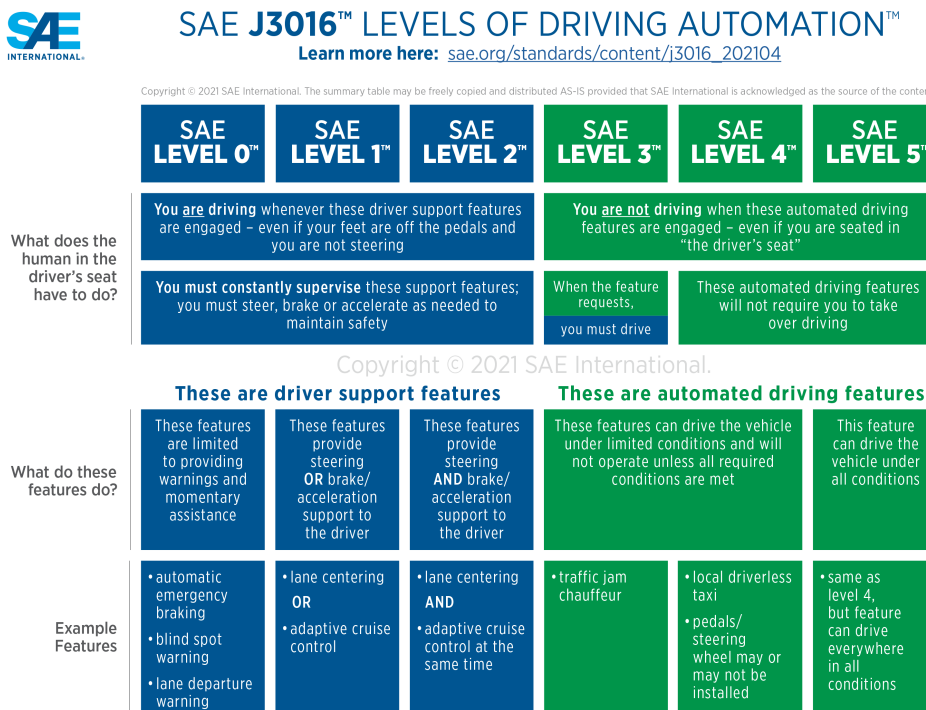


Figure 1: Proposed levels of AD according to [2].

As mentioned before, **AD** encompasses several mechanisms, some simpler, others more complex. Usually, the latter are the only ones considered as **AD** by the general public. With that said, nowadays, there is a great desire to reach the level of complete autonomy, due to the safety it provides, and to increase the population's trust and endorsement in this technology, as it is in everyone's interest to improve road safety and road traffic.

The notion of **AD** has been around for many years, although it was brought to light again in 2004 by DARPA [21], creating a challenge to accelerate the development of **AV** technologies, where no team was able to complete it. A year later, in 2005, another round was held, but this time 5 vehicles completed the proposed course. These challenges contributed to the creation of the required mindset, and in the next decade multiple companies started researching and investing in this field. In 2009, *Google* started a research project on **AV**, subsequently creating the *Waymo* project. Nowadays, **AD** is one of the most researched areas in the automobile industry.

A typical **AV** consists of multiple modules, from perception to vehicle controls, that work together to make decisions. Thus, to drive safely, an **AV** must analyse data streams generated by their sensors in order to understand their surrounding environment and use this knowledge to make the best informed driving decision, making perception a crucial task for **AD**. This dependence on scene understanding requires this perception task to be done accurately, because otherwise it can result in potential mistakes, possibly resulting in fatalities. Therefore, the quality of said perception models are of the utmost importance.

Given the complexity behind these technologies and the recent progresses in many fields regarding Deep Learning (**DL**), such as object detection and Multi-Object Tracking (**MOT**), current perception systems benefit from the use of **DL** approaches to be executed, even in complex and highly dynamic scenarios.

1.2.2 Perception on AI (LiDAR sensors)

Sensors are essential to ensure this task is performed with a degree of safety. The most common types of sensors in this field are colour (**RGB**) and Infrared (**IR**) cameras and depth sensors (including **LiDAR** and radar). A normal camera (**RGB** or **IR**) has the advantage of being cheaper and its data is presented in a more common format, making development easier. Nonetheless, it lacks depth precision and it has poor performance under extreme weather conditions. On the other hand, depth sensors allow for a more precise understanding of the scene, providing the objects' location, how far apart they are, and, in most cases, 360° of visibility, which allows an **AV** to create a 3D map of its surroundings. However, depth sensors are more expensive, provide much sparser information, which is amplified for a distant object, and require additional space and resources to be integrated on vehicles. Nevertheless, these two types of information can be used together for an improved spatial representation of its surroundings. This environment understanding comprises several perception tasks, in which the ability to detect and track objects is key for **AV**.

While keeping track of objects is a relatively easy task for a person, for a computer, on the other hand, it has been not so simple. Nonetheless, many have been researching and pushing boundaries of computer's performance for

this type of task. Recent techniques have brought clear improvements, making tracking feasible to be applied in the context of AD.

LiDAR is a method for measuring the distance between the sensor and the multiple obstacles by emitting lasers beams and calculating how much time the reflected light takes to return. The idea behind it is similar to radar but instead of radio waves, it uses light waves. This technology is extremely accurate for object distance measurement, even up to millimetres.

Recently, AV started using **LiDAR** [22] since it provides a 3D point cloud representation that describes its surroundings. As mentioned previously, the main advantage of this type of sensor is its accurate measurement regarding distances, which is useful to improve scene displacement and mitigate obstacle occlusion and avoidance. However, the choice of sensors differs across companies, while some opt for a **LiDAR** approach, others choose to invest in traditional vision-based systems. While most companies conducting research in AV use **LiDAR**, *Tesla*, heavily relies on computer vision, using mostly cameras. On the other hand, other well known companies, e.g., *Uber*, *Waymo*, and *Toyota*, all use **LiDAR**.

1.2.3 Artificial Intelligence

Artificial Intelligence (**AI**) is the ability to perform “intelligent” tasks by machines. This field was “born” in 1956 at Dartmouth College [23]. In the following years, research increased and led to inspiring results. However, this progress slowed down, and eventually in 1974, funding was withdrawn, which made **AI** face its first “winter”, due to limited computer resources and the complexity problems at hand.

Almost a decade went by before any major breakthrough occurred until, in the early 1980s, **AI** was revived by the success of expert systems [23]. In spite of that, in 1987, **AI** fell into its second “winter” [24] due to unattainable expectations which lead to a decrease in funding and the loss of interest. Many experts claim that this happened due to these systems’ lack of common sense and knowledge about their own limitations [25].

By the turn of the new millennia, **AI** was brought to light once again through the research of findings to solve specific complex problems (e.g., *Deep Blue*, a computer chess-playing system, beat the reigning world chess champion, Garry Kasparov [26]). This narrow methodology allowed researchers to benchmark algorithms’ results, as well as apply these methodologies to other fields [23]. However, these progresses did not occur by any breakthroughs, instead they were achieved due to better hardware, faster computers and improved capacity.

At the moment, many applications incorporate **AI** solutions, with its growth being driven by many factors, such as access to large amounts of data (“big data”), and access to cheaper, mobile and faster devices. These progresses were also propelled by advances in **DL** approaches and other techniques.

AI has become a part of our daily life, as it is in our pockets via cell phones and personal assistants. This rise is due to its advantages, such as: (i) excel at detail-oriented tasks; (ii) fast at data-heavy tasks; and (iii) availability,

since AI agents are available 24/7. However, it also presents some disadvantages, including: (i) inability to adapt to new scenarios; and (ii) disability when generalising tasks, i.e., transferring knowledge from one task to another. AI can be divided into the following sub-categories, as presented in Figure 2:

- *Symbolic Artificial Intelligence* represents AI methods that are based on high-level symbolic representations (i.e., human-readable), such as rules and inference mechanisms;
- *Evolutionary Algorithms* are inspired by the natural biological evolution and are usually used to optimise solutions. This procedure follows the standard structure:
 1. Generate a random set of individuals (solutions);
 2. Compute the fitness (quality metric to the objective) of each individual in the population (set of solutions);
 3. Apply multiple genetic adaptation operations into a set of individuals, to generate a new breed of individuals (solutions);
 4. Repeat steps 2. and 3. until a desired individual (solution) is reached.
- *Machine Learning (ML)* refers to algorithms that automatically improve through the analysis of recorded past experiences and data consumption.
 - *Artificial Neural Networks (ANN)* is one of the most relevant field of ML. This type of architecture was inspired by the human brain, trying to recreate the brain's reasoning behaviour.
 - * *Convolutional Neural Networks (CNN)* is a variation of an ANN, although it applies a technique to process its data, and, usually, it is applied to matrix information, such as images;
 - * *Deep Learning (DL)* is a “deeper” ANN, usually, is better at solving more complex tasks, such as pattern and speech recognition, AV, and more.

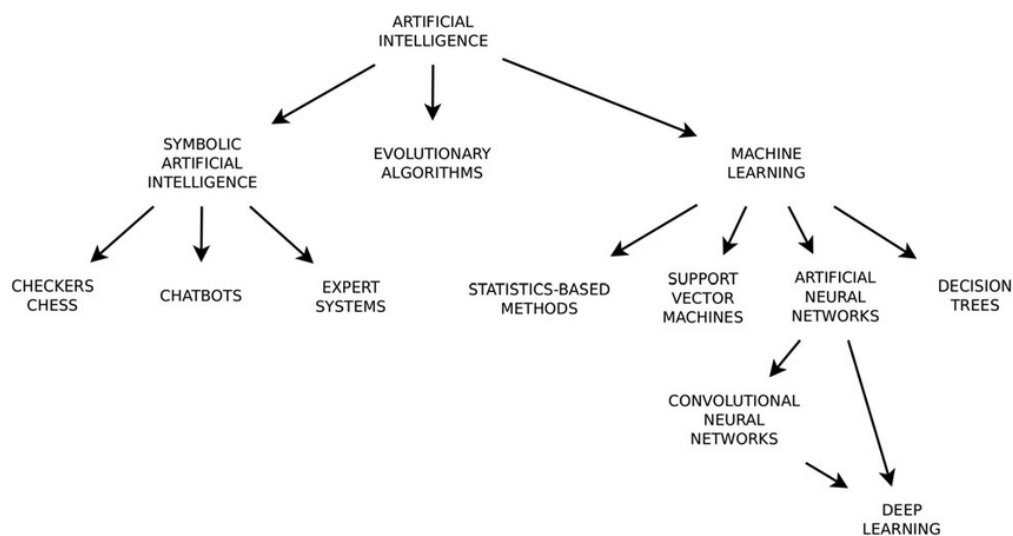


Figure 2: Overview of the main branches of AI, extracted from [3].

In this dissertation, the focus will be incised on DL, more specifically Artificial Neural Network (ANN).

1.2.4 Machine Learning

Machine Learning (ML) is a subdivision of AI. Its goal is to study/develop computer algorithms that automatically improve through experience and data consumption [27], without being explicitly programmed to do so. It consists on developing learning models based on samples, known as training data, to make predictions/decisions based on previous use cases, i.e., data.

Within ML, there are three main model learning methodologies applied to its training:

- *Supervised learning*, is defined by the fact that the algorithms have access to an input and its expected output, representing use cases experiences for a specific problem to be solved. As the input data is fed into the model, it adjusts its weights. This learning is based on the idea that if the model knows how to classify/predict for previous/sample data, it can also predict for new/real data. To prevent the model from getting too reliant on the training data (overfitting), or to ensure that it can capture the relationship between input and output (underfitting), some techniques are applied (e.g., cross-validation, where data is divided into training, validation and testing, to evaluate the performance of a model);
- *Unsupervised learning*, training data is composed of unlabelled data. Each sample has features, and the model tries to discover hidden patterns to group samples according to similarity. This type of learning is characterised by the fact that it does require human intervention since no annotations are present;
- *Reinforcement learning*, instead of using input data, the model receives feedback from an agent operating in a controlled environment. The mentioned feedback is the result of the outcome of its actions or the proposed output from the model, resulting in a reward given by the environment, culminating in a trial and error process. The model aims to maximise its total reward, with encouragement to achieve the best possible outcome.

In addition, there are more variants:

- *Transfer Learning*, it relies on previous pre-trained learning models trained for similar scenarios, and uses them as a starting point to shorten the training process to solve a similar problem;
- *Semi-supervised learning*, is a mix between supervised and unsupervised learning, where a small amount of data is labelled, while the rest is unlabelled.

1.2.4.1 Deep Learning

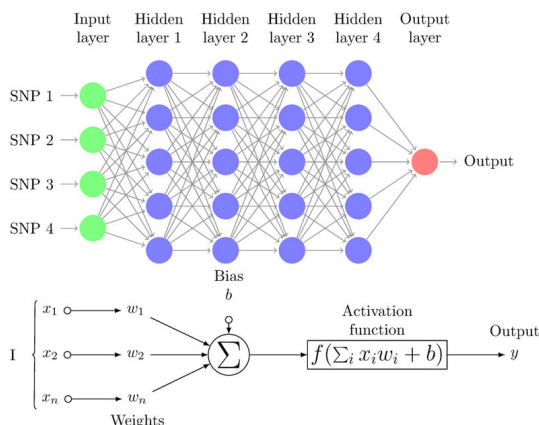
Deep Learning (DL) is a subset inside ML, and it is characterised by the application of ANN to simulate the behaviour of the human brain [28]. A ANN with more than 3 layers (input, output and 2 or more hidden layers) is called deep neural network, hence the “Deep” in DL.

DL algorithms have presented some advantages such as their flexibility for solving many types of problems and normally present good results when training with large amounts of samples.

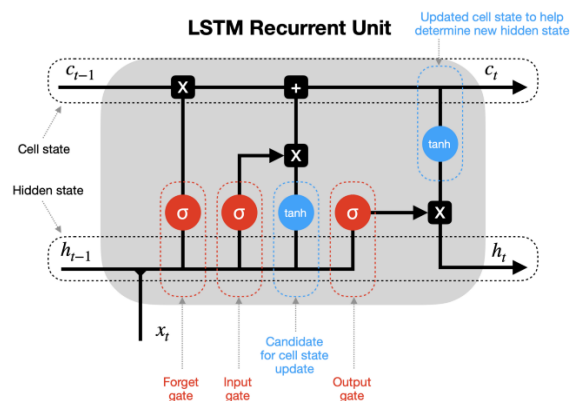
DL is relevant across multiple essential tasks in AD, such as perception, planning, and control. When it comes to perception, DL is applied in many tasks, including lane detection, obstacle detection, and distance estimation.

There are multiple types of DL algorithms. The most relevant networks are:

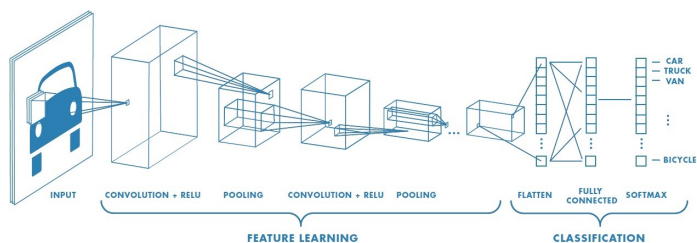
- Multi-layer Perceptron (MLP) belong to the class of feed-forward neural networks with multiple layers of perceptrons. Its inner workings will be described in section 1.2.4.2. These networks are simple in terms of organisation, although their size rapidly grows. Its main applications are speech recognition and image recognition. An example of a MLP is illustrated in Figure 3a;
- Convolutional Neural network (CNN), also known as ConvNets, was first developed in 1988, and is a variation of the MLP networks. Its main difference is the use of convolution layers, which is the main building block. This layer contains a set of filters, weights which are to be learned through training. Each filter convolves with the input and creates a feature map. These layers are normally applied to matrix information, such as digital signal and image processing. Pooling layers are usually applied together with convolutional layers, this layer reduces the dimensions of data. Typically, the two most common types of pooling are Max and Average. As the name suggests, max pooling calculates the maximum value for patches of a feature map, while average pooling calculates an average for each patch of a feature map. Its main applications are image processing and object detection due to their capacity for image recognition and identifying different image patterns. An example of a CNN is illustrated in Figure 3c;
- Recurrent Neural Networks' (RNN) structure resemble a MLP network, however its connections are not all feed-forward, i.e., the output of a given node is a node in a previous layer which creates cycles. This type of algorithm is useful in maintaining a small state which is very useful for developing chat-bots and text-to-speech applications. An example of a RNN is illustrated in Figure 3d;
- Long Short Term Memory (LSTM) Networks are a variant of RNNs. These networks can learn and memorise long-term information by remembering data over time. It has a chain-like structure where four interacting layers communicate. A standard LSTM unit contains a cell, an input, an output and a forget gate. The cell is responsible for retaining knowledge over an arbitrary time while gates control the flow of information into and out of the cell. The primary purpose of a LSTM is to forget the irrelevant information of prior states and selectively update its current state while retaining relevant information. Usually, LSTM networks are used in temporal contexts, such as classifying, processing and predicting data from time series due to their memorisation abilities. An example of a LSTM is illustrated in Figure 3b.



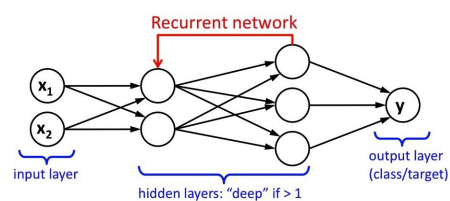
(a) Example of a MLP network, extracted from [29].



(b) Example of a LSTM network, extracted from [30].



(c) Example of a CNN network, extracted from [31].



(d) Example of a RNN network, extracted from [32].

Figure 3: Representation of multiple types of DL architectures.

Other types of DL architectures are also proposed in the state-of-the-art, e.g., Generative Adversial Networks (GANs) which specialise in generating new data, and Autoencoders, which are suitable to detect anomalies. However, due to the short time of this research, these will not be considered for the problem at hand.

1.2.4.2 Artificial Neural Networks

Artificial Neural Network (ANN)s are inspired by the human brain and try to mimic its behaviour, as explained in [33]. These networks are composed of multiple node layers as illustrated in Figure 3a: (i) an input layer, which receives the data; (ii) an output layer, which returns the result; and (iii) one or more hidden layers. Each of these layers is composed of one or more nodes and connects to nodes from the previous and following layers through channels, which have weights that dictate the final result.

A Neural Network operates by, as explained in [34]:

- Initialising the network weights, usually between 0 and 1, where these are defined based on the weight initialisation technique applied (e.g., random weight initialisation);
- Training the model with four steps:
 1. Feed data to the input layer;

2. Each node receives a value from the preceding layers. After, it calculates its output value using the incoming values and their respective channel's weight. Finally, the neuron adds a bias to this value, and it is filtered using the activation layer and then transmitted to the subsequent nodes. This process is illustrated in Figure 4b;
 3. When it reaches the output layer, the network compares its output with the original result to calculate the error offset, i.e., how close it is to the expected result.
 4. Apply the output's difference to adjust its weights to propagate it to the previous layers, minimising the overall error.
- The model repeats these four steps until it reaches an acceptable performance or exhausts the pre-defined epochs.

These two propagation processes (transmitting data to the following layers and propagating the error to previous nodes) are fundamental to training an ANN.

- *Forward propagation* is the most used method of transmitting information from the input layer to the output layer, i.e., equivalent to transferring data from the left nodes to the right ones in Figure 4a. This operation corresponds to steps 1 and 2 and is used to obtain a possible result.
- *Backward propagation* is a method of adapting weights to reach a minimised loss function and is comparable to transmitting information from the right to the left layer in Figure 4a. This procedure corresponds to steps 3 and 4 and its goal is to improve the overall performance.

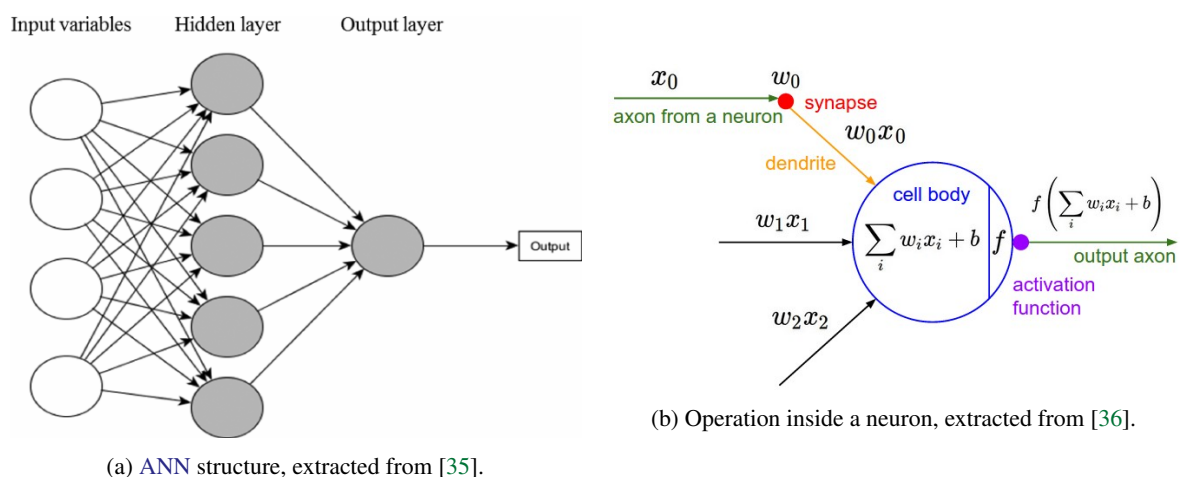


Figure 4: Network and Neuron representations

Finally, this optimisation is performed using an algorithm called gradient descent. It is an optimisation algorithm used to find a local minimum of a differentiable function. The idea behind this is that enough steps in the opposite direction of the gradient (which indicates the direction where the function locally grows faster) will, eventually, lead

to a local minimum, as illustrated in Figure 5. In machine learning, this technique is used to minimise the loss of a model, by updating its parameters. This gradient is calculated after each step to obtain the correct gradient, otherwise, it would always move in the same direction and would not ensure the minimisation of the function. This process usually ends when it can no longer move to a lower point.

The learning rate influences this process by dictating the size of each step, a high learning rate moves faster, although it is less likely to reach the lowest point. On the other end, with a low value, the model is more precise and will move in the “right” direction due to calculating the gradient so frequently, however, it will take longer to reach the minimum. [37]

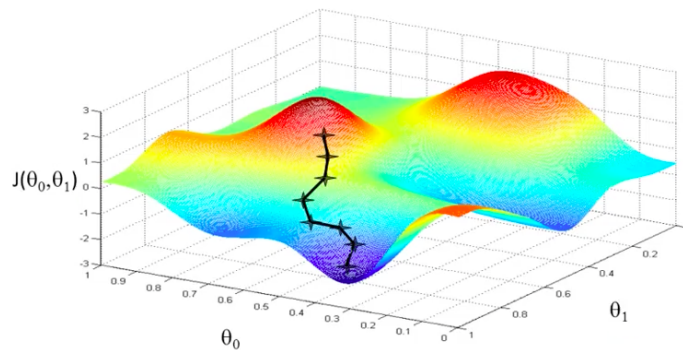


Figure 5: Illustration of the gradient descent algorithm steps to minimise the cost function, extracted from [4].

The update function for each node in the neural network is described in Equation 1.

$$W'_i = W_i + \alpha \cdot \nabla E \quad (1)$$

where W'_i and W_i are the new and old weight, respectively. α is the learning rate, E represents the error. The error can be obtained through the equation 2.

$$E = \frac{1}{2}(p - gt)^2 \quad (2)$$

where p and gt correspond to the prediction and the actual value, respectively. It worth noting that the power of 2 (2) forces the error to be positive in every situation.

1.2.5 Object Detection

Object detection is a key technology (applied also in AV) that can be defined as an image processing and computer vision task that aims to locate and classify objects in image and video data. Generally, it uses machine learning and/or deep learning algorithms (using supervised learning), to perform such task [38].

Although its similarity to image classification (which assigns a classification label into an image), object detection entails two tasks: (i) object localisation (e.g., labelled via a bounding box, presenting the location, size and orientation of the object); and (ii) classification (class/category of the detected objects).

Object detection works for either 2D or 3D information, with some key distinctions. While in 2D the algorithm is fed with image frames, in 3D, usually, the input data refers to a 3D point cloud. In terms of 2D detections, the output is a set of axis-aligned bounding boxes with, generally, two coordinates, while in 3D, the output is a set of 3D bounding boxes, with slight variations from model to model (e.g., it can be an object's centre and its dimensions or a bounding box vertex and its dimensions).

As mentioned above, an unbiased evaluation is a key factor in ML. The basic concepts associated with evaluation are the following:

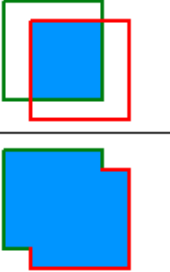
- True Positive (TP), represents a correct detection, i.e., an object is detected and associated with a ground truth object;
- False Positive (FP), represents a false detection, i.e., an object is detected but it does not correspond to any ground truth;
- False Negative (FN), represents a missed object, i.e., a ground truth object is not associated with any detection;
- True Negative (TN), for this type of task, it is impossible to measure, however, TN corresponds to the image regions that have no ground truth objects and detections.

Furthermore, these are the most used evaluation metrics in object detection:

- Intersection over Union (IoU) is a standard metric applied to evaluate this type of algorithm. In short, this metric measures the intersection of two geometric shapes divided by the overlapped area between that same shapes, as illustrated in Figure 6 and described in equation 3, with “p” and “gt” representing respectively predicted and ground truth bounding boxes.
An IoU of 1 implies that the detection and ground truth bounding boxes are the same, i.e., a perfect overlap. Regarding 2D object detection, “p” and “gt” correspond to areas, while in 3D, the calculation is done with volumes;
- Precision (equation 4) measures how accurate predictions are. In object detection, it can be interpreted as the percentage of detections that were correctly made;
- Recall (equation 5) measures the percentage of positives that were correctly classified, i.e., the percentage of objects that were detected;
- Precision-Recall Curve measures the tradeoff between precision and recall for different thresholds. A high value represents both high recall and high precision, and while high recall relates to a high percentage of

found objects, and high precision to a low percentage of wrongly detected objects. Therefore, a high value indicates high values for both, returning accurate results, detecting a high amount of ground-truth objects;

- Average Precision (AP) and mean Average Precision (mAP) stands for average precision and mean average precision, respectively, as shown in equations 6 and 7. AP summarises the precision-recall curve into a single value, it can be viewed as the area under the curve, or, mathematically, as the sum of the precision values for each threshold. While mAP is the average AP for every class.

$$IoU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{area of overlap}}{\text{area of union}}$$


$$Pr = \frac{TP}{TP + FP} \quad (4)$$

$$Re = \frac{TP}{TP + FN} \quad (5)$$

Figure 6: Illustration of IoU, taken from [5].

$$AP = \sum_{r \in R} p(r) = \sum_{r \in R} (Re_r - Re_{r-1}) Pr_r \quad (6)$$

$$IoU = \frac{TP}{TP \cup FN \cup FP} \quad (3) \quad mAP = \frac{1}{L} \sum_{cinCs} AP_c \quad (7)$$

where p and gt are the prediction and the ground-truth, respectively. Re_r and Pr_r are the recall and precision value for threshold r , respectively.

1.2.6 Tracking for Object Detection

Object tracking, more specifically Multi-Object Tracking (MOT), is an algorithm that receives, usually for every frame, a set of object detection outputs and assigns to each of them a unique identifier. However, if one object is already being tracked, it should assign that same identifier across all frames. This way, the algorithm stores for each tracked object (known as target) its state information (known as track), keeping it up-to-date, providing the conditions for an high accuracy of its trajectory. Although it is more common to associate MOT with RGB images and video, it works with any type of temporal consistent input, in this dissertation's case, using LiDAR's 3D point cloud.

Even though MOT is closely related to object detection, most approaches in the past treated MOT as an extension to the base algorithm, without taking advantage of DL algorithms. Recent AI approaches brought significant improvements for the overall performance.

There are two main types of object tracking:

- **Single Object Tracking (SOT)**: receives the first frame with bounding boxes, identifying its only target. This type of tracking can be applied even in dynamic environments with other objects. After the initial frame, the tracker is, then, responsible for locating that unique target across all next frames;
- **Multi-Object Tracking (MOT)**: is responsible for identifying and tracking multiple objects. This type of tracking is applied to **AD** where its ability to detect and predict the behaviour of other vehicles or pedestrians is essential for decision-making.

As said, **MOT** is crucial for **AV** and **AD**, therefore, this will be this section's main focus.

The two most common frameworks of **MOT** are the following:

- *Tracking-by-detection paradigm*: essentially there are two main modules - detection and tracking. In this type of approach, the object detector is independent from the rest of the **MOT**, i.e., multiple detectors can be used to detect objects, as long as they output the same type of information. The rest of the algorithm is responsible for the tracking tasks, such as association between already tracked objects and detections, prediction of future state, e.g., trajectory, velocity, initialisation of new tracks and termination of no longer in range objects. This approach has the advantage of being well structured and its modules are independent from each other, however, it heavily relies on the performance of the detector [39];
- *Joint detection and tracking paradigm*: this approach has a much simpler description. Here, detection and tracking are no longer divided, and work as a single module. The advantage of this approach is its optimisation, since detection is trained to increase algorithm's performance, instead of its own. This leads to better results, however, the complexity in this type of framework increases drastically.

Every **MOT** has two possible approaches:

- *Online*, where the **MOT** can only use present and past information to perform tracking for each frame. However, this mode does not imply that tracking is done in real-time - in **MOT**, real-time means that the algorithm returns its output before the next frame. Usually, the threshold for real-time is 10 frames per second (Frames-per-second (**FPS**)), since it is the standard frame rate for **LiDAR** sensors. This type of approach is used, mostly, in real-time applications, such as **AD**, yet leads to lower performances due to using only past information;
- *Global*, in contrast, uses all available information to perform its sub-tasks, e.g., maximise association performance for the whole sequence instead of what has already been used as input. This type of approach leads to higher performances due to using future information, which invalidates its usefulness in real cases where the future is unknown.

As previously mentioned, tracking-by-detection applies the concept of association. Usually, this task is performed by quantifying the similarity between detections and targets to pair them, resulting in a match. Normally, it is used **IoU**, statistics methods/formulas and/or distance between tracked and detected objects' centres. Others use more complex methodologies, e.g., based on geometric and appearance (usually uses RGB data), or **DL** algorithms. Nonetheless, this comparison is not made with the stored/current information, since this information is relative to

the previous frame (i.e., there is an offset of one frame between detections and the said information). To mitigate this delay, the algorithm tries to predict the next frame state, i.e., predict the position where each objects will be, their velocity, and orientation, among others.

After computing all matches, it is necessary to decide which new tracked objects should be considered as a new track, and what missing objects should be dismissed to stop its tracking. Generally, there are two ways to tackle this problem - predefined, i.e., the number of necessary consecutive frames to start tracking a new object and stop tracking missing objects; DL approach, i.e., training this module to decide when to start/stop tracking a given object.

Given that MOT is applied to AD, detection is not made from a stationary position, with ego-motion emerging as a problem since the detector is only capable of returning relative information. Therefore, some techniques are used to mitigate this, such as GPS integration, to convert information from relative to absolute, or tracking with only relative information. Nevertheless, this approach magnifies the measurement uncertainty.

In the next sections, we will focus on analysing the current state-of-the-art solutions regarding MOT on AD, and identify its main advantages and limitations.

1.3 DOCUMENT STRUCTURE

This document will consist of several chapters with the following content:

- Section 2 - State-of-the-art: presents an overview of several state-of-the-art researches, their stated results, and a list of their main advantages and disadvantages for the task at hand. It will conclude with a brief discussion and benchmark of the considered works;
- Section 3 Objectives: a list of objectives is proposed, what is expected and what are the main targets for this dissertation, as well as the definition of tasks to achieve the expected goals;
- Section 4 - LiDAR-based object tracking: consists of an overview of the available datasets, followed by a more in-depth analysis of the chosen dataset and the data pre-processing techniques for the task at hand. Then, the detection model is introduced briefly, and a initial architecture is proposed along with its expected output. Followed by the dataset pre-processing and data augmentation are discussed and introduced, and the current architecture is explained with some more-in-depth analysis for the major architecture's components. Finally, the evaluation metrics to benchmark the results across different architectures are explained, ending with the hyper-parameter tuning process. Lastly, an overview to summarise the most relevant information in this section;
- Section 5 - Experiments and Results: in this section, the evaluation process is described, followed by the quantitative and qualitative performance evaluation. Finally, a analysis of the resources and a ablation study is performed;
- Section 6 - Conclusions, Limitations and Future Work: a brief conclusion of the research work, presenting an overview of the results produced, definition of the current limitations and the proposal of next steps to be achieved in the future.

STATE-OF-THE-ART

The main goal for this scientific exploration is to analyse the most used datasets for benchmarking for MOT solutions and its characteristics, the results achieved by each, to assess the most common approaches, pipelines and techniques for data pre-processing, data association, track initialisation and deletion, as well as the most common limitations and obstacles.

2.1 BEYOND PIXELS

With the emergence and subsequent surge in AD, accurate MOTs become essential for several tasks, such as navigation and planning, localisation, and traffic behaviour analysis. This paper [6] focused on designing a simple and fast, yet accurate and robust MOT applied to urban road scenarios, by taking advantage of objects' shapes and poses temporal-consistency.

In 2018, Sharma et al. [6] brought major improvements to MOT. The proposed architecture is characterised by using a tracking-by-detection approach and taking advantage of some characteristics inherent to the object and its movements, such as object pose, shape and motion. To do so, it uses both LIDAR and RGB data.

Since the algorithm employs a tracking-by-detection approach, the first step is to send the corresponding frame to the detection algorithm. Then, using its output (in this case, a set of detections, each with the top-left corner coordinates, width and height of the bounding box, as well as the confidence), it computes the association cost for every object. As described previously in section 1.2.6, association cost is a measure of the similarity between detection and tracks. However, instead of the traditional costs, the following are calculated:

- *3D-2D cost* measures the overlap of each current detection proposal's 2D region with the all 3D detections projected into the 2D plane from the previous frame, as illustrated in Figure 7;
- *3D-3D cost* measures the overlap of each current detection proposal's 3D region with the all 3D detections from the previous frame, as illustrated in Figure 7. In reality, it is assumed all bounding boxes have the same height, and, therefore, the comparison is made in bird's-eye view;
- *Appearance cost* uses a deep learning neural network (Deep Learning Neural Network (DLNN)) to measure appearance between 2D regions;

- *Shape and Pose cost* explores additional information provided by an external algorithm that computes a vector of deformation coefficients. With that information, it calculates the similarity between each detection and all tracked objects.

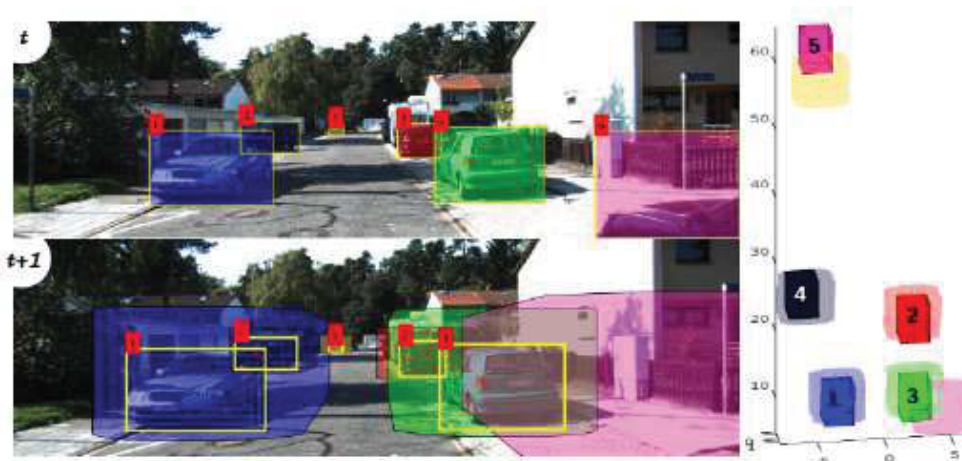


Figure 7: Concept of 3D-2D and 3D-3D costs using 2 consecutive frames on the left and right respectively, extracted from [6].

After all the necessary costs are computed, a hungarian algorithm [40] is used, to associate new detections and tracked objects.

The proposed approach is a tracking-by-detection and hence assumes a set of detections as input, repeating this process for every frame. This work uses two different object detectors to compare results; then, detections are filtered by applying a threshold for confidence scores; and, additionally, a non-maximum suppression (NMS) is used to reduce multiple detections around the same object. Finally, pairwise costs are computed to associate pairs using a matching algorithm, i.e., the aforementioned hungarian algorithm.

In terms of results, this paper achieved state-of-the-art performance with a multi-object tracking accuracy (Multiple Object Tracking Accuracy (MOTA)) of 84.24% on the KITTI dataset [1] as is presented in Table 1.

Despite its good results, this method has a high number of identity (ID) switches and can only run at 3 FPS¹, which is far from the 10 FPS threshold of real-time tracking. Even though Sharma et al. states that this problem (ID switches) is the result of the proposed approach implementing an online tracker, it can be mitigated by using a tracker with a more sophisticated cost association.

¹ According to http://www.cvlibs.net/datasets/kitti/eval_tracking.php

	MOTA	MOTP	MT	ML	IDS	FRAG
CIWT	75.39	79.25	49.85	10.31	165	660
SCEA	75.58	79.39	53.08	11.54	104	448
MDP	76.59	82.10	52.15	13.38	130	387
LP-SSVM	77.63	77.80	56.31	8.46	62	539
NOMT	78.15	79.46	57.23	13.23	31	207
MCMOT-CPD	78.90	82.13	52.31	11.69	228	536
Ours(RRC-IIITH)	84.24	85.73	73.23	2.77	468	944

Table 1: Beyond Pixels results on the KITTI test set [1], extracted from [6].

As presented in Figure 8, this system works well in a scene with multiple objects, even for faraway objects, as it is able to track them without any hurdle. Furthermore, it succeeds in tracking partially occluded objects, like the vehicle in the right column frames, tracking it until it completely disappears.

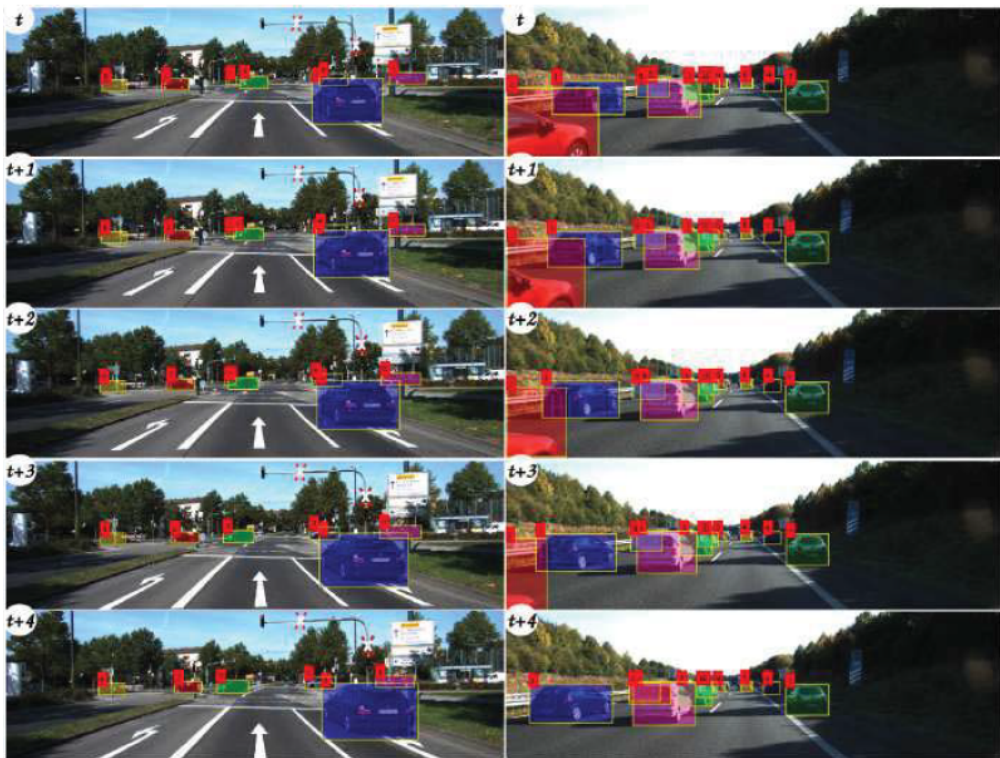


Figure 8: 2 Beyond Pixels tracking examples and respective results, extracted from [6].

2.2 FANTRACK

In 2019, [Baser et al. \[7\]](#) used two siamese networks to model the similarities between tracked objects and detections, as well as DLNN to solve the association problem in MOT, represented in Figure 10.

Here, multiple branches inside the similarity network for the association cost problem are applied, as illustrated in Figure 9:

- *Bounding box branch*: outputs a vector of dimension features for the current detections and the targets (i.e., tracked objects), later used to calculate bounding box similarities;
- *Appearance branch*: outputs a vector of 2D visual cues for both detections and targets;
- *Importance branch*: receives as input the first two branches, and its goal is to determine the relative relevance of each branch.

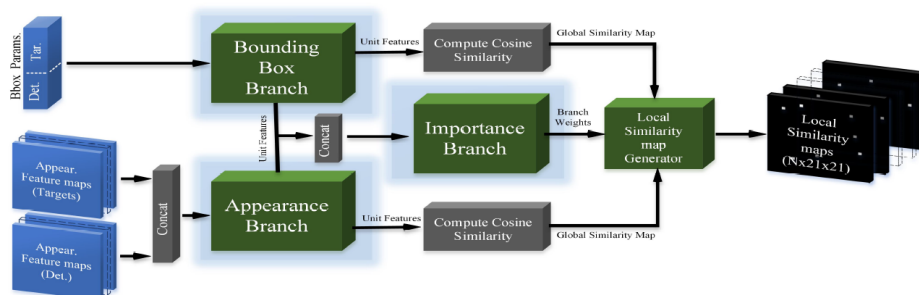


Figure 9: FANTrack Similarity Network Pipeline, extracted from [7].

Afterwards, it generates a similarity map for each target, where the outputs of the previous branches are considered, i.e., the weight of the first two branches (bounding box and appearance) is set by the latter branch (importance). The next step is to associate targets with detections to create pairs. This is done once more by a siamese network that outputs the probability for each target-detection-pair.

Finally, the tracking information is updated, a future state prediction is made for each target, and some tracks are initiated and pruned.

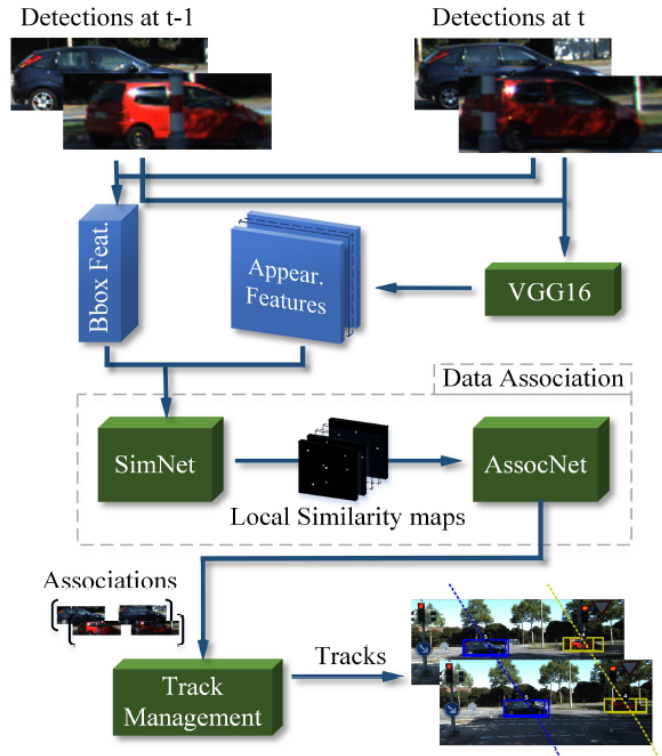


Figure 10: FANTrack proposed system pipeline, extracted from [7].

In terms of results, as presented in Table 2, this paper achieved a worse performance (MOTA of 77.72%) than Sharma et al.'s approach, however, it had fewer ID switches and ran at a higher rate (25 FPS). It is also penalised by the KITTI evaluations because they are done in 2D, while this approach provides inferences in 3D. Track management, future state and motion prediction presented classical approaches, leaving room to improve these results.

Method	MOTA \uparrow	MOTP \uparrow	MT \uparrow	ML \downarrow	IDS \downarrow	FRAG \downarrow
MOTBeyondPixels	84.24 %	85.73 %	73.23 %	2.77 %	468	944
JCSTD	80.57 %	81.81 %	56.77 %	7.38 %	61	643
3D-CNN/PMBM	80.39 %	81.26 %	62.77 %	6.15 %	121	613
extraCK	79.99 %	82.46 %	62.15 %	5.54 %	343	938
MDP	76.59 %	82.10 %	52.15 %	13.38 %	130	387
<i>FANTrack (Ours)</i>	77.72 %	82.32 %	62.61 %	8.76 %	150	812

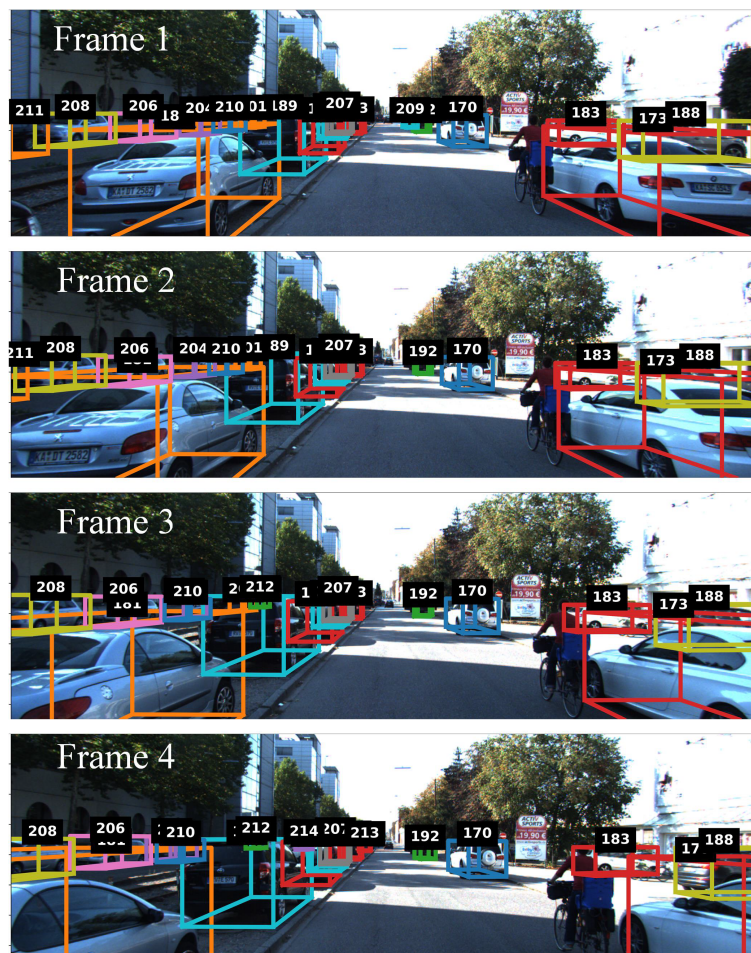
Table 2: FANTrack results on the KITTI test set [1], extracted from [7].

Once more, the KITTI tracking benchmark dataset was used for training and evaluation purposes, consisting of 21 training sequences (20% of every training sequence was used for validation, therefore no bias was introduced)

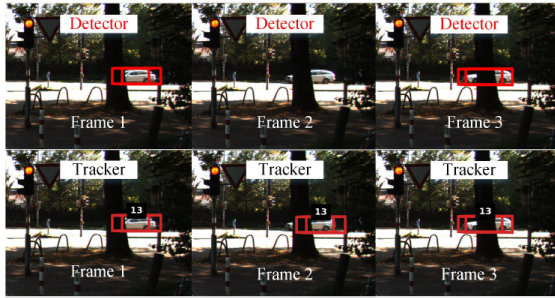
and 29 test sequences. This distribution was considered during the research of this work.

For the training of the similarity network, the researchers prepared a dataset from the training sequences by generating positive and negative (ratio of 25 to 18) examples in consecutive frames using manually labelled information. Geometric transformations, such as translation, rotation, and scaling, were applied to the ground-truth data to recreate partial occlusion and detector noise. The object detector was trained with a combined dataset consisting of the KITTI 3D object detection dataset and the 80% split of the KITTI training dataset mentioned earlier after pre-training on a synthetic dataset.

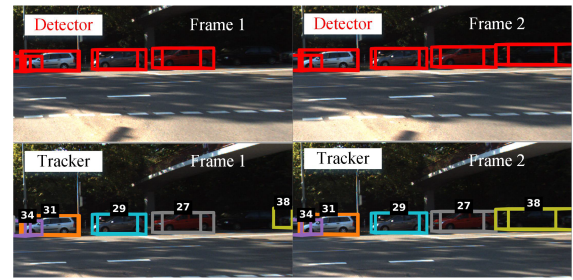
As we can see in Figures 11b and 11a, this algorithm performs well in a cluttered scene and is capable of tracking partial occluded or missing objects, even when the detector fails to detect them. However, due to the poor lighting conditions presented in Figure 11c, the algorithm assigns an incorrect ID, resulting in an ID switch.



(a) FANTrack cluttered example, extracted from [7].



(b) FANTrack missed detection recovery example, extracted from [7].



(c) FANTrack ID switch example, extracted from [7].

Figure 11: FANTrack tracking examples

2.3 AB3DMOT

In 2019, [Weng and Kitani \[8\]](#) proposed a fast and effective 3D MOT, as well as new metrics for evaluation, and a 3D MOT evaluation tool. This architecture, unlike others, only uses 3D data, and it presents positive results compared to its peers.

In terms of architecture, it has a simple and well-structured pipeline that is divided into several modules, as presented in [Figure 12](#):

- *Object detector*: responsible to detect each objects' position, orientation, size, as well as confidence related to that detection;
- *State prediction*: a Kalman filter that predicts objects' next frame state (position, direction, and velocity). Note that the detection does not output velocity. Instead, it is only obtained after the second frame. With that in mind, [8] found, empirically, that angular velocity does not provide improvements, *per se*, therefore, a constant velocity model, updating only its position ($p_t = p_{t-1} + v$), was implemented;
- *State update*, to account for detection and tracking uncertainty, object data is updated with a weighted average of both. Nevertheless, this method does not work well for objects' orientation, since, through measurement error, it can almost have opposite angles resulting in an even worse angle. Thus, for cases where the difference between both angles is greater than π , they decided to add π to the tracking angle, solving any abrupt change;
- *Data association*: done by a hungarian algorithm, rejecting every pair that has a IoU lower (or a centre's distance higher) than a pre-defined threshold. This results in 3 sets: tracks and detections matched (found and tracked objects), tracks unmatched (tracked objects missing), and detections unmatched (possibly new objects). These sets are then used to perform state updates as described above;
- *Birth and death memory*: this module is responsible for starting and deleting tracks. Its main goal is to avoid creating false positives (since an unmatched detection could be an error from the detector) and false

negatives (since the detector could miss an object due to, for example, poor lighting). Therefore, a new track is only created if that same new object is detected a pre-defined number of times and is only deleted if it has not matched with any detection for a pre-defined number of times.

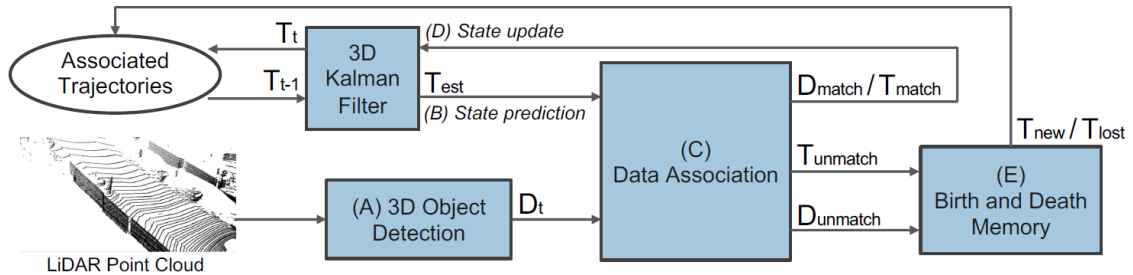


Figure 12: ABD3DMOT proposed pipeline, extracted from [8].

Regarding the proposed metrics, [Weng and Kitani](#) claim that conventional MOT evaluation is based on metrics, e.g. MOTA, that do not consider the confidence score for a track. To reduce the number of false positives and achieve a higher MOTA, it is necessary to manually select the desired threshold. However, this selection is a non-trivial and slow task. Apart from that, this threshold is dependent on the object detector and on the fed data, differing from dataset to dataset. Finally, one more problem arises due to using only a single threshold, preventing us from getting a global understanding of the 3D MOT system.

With that in mind, a set of new metrics is proposed, being average multiple object tracking (Average Multiple Object Tracking Accuracy (AMOTA)) (explained later in section 4.8) the most relevant. These metrics are so relevant and hence, are still the standard in some datasets, e.g., applied in [16].

For this research, the KITTI 3D tracking dataset was used, splitting it into validation and training sets (with a custom selection), even though this system does not need training. For the nuScenes dataset, the default split was used.

In terms of results, this paper achieved state-of-the-art 3D MOT performance on the KITTI dataset, across multiple IoU thresholds. For the class “car”, it outperformed other 3D MOT systems in all metrics, as presented in Table 3. It also achieved an impressive 0 ID switches and 207 FPS (CPU based). For “pedestrian” and “cyclist” classes, it presented worse results due to its smaller dimensions. These results are shown in Table 4. They also performed benchmarks using the nuScenes dataset, and, once more, it outperformed both analysed systems, as presented in Table 5.

Method	Input Data	Matching criteria	sAMOTA \uparrow	AMOTA \uparrow	AMOTP \uparrow	MOTA \uparrow	MOTP \uparrow	IDS \downarrow	FRAG \downarrow	FPS \uparrow
mmMOT (ICCV'19)	2D + 3D	IoU $_{\text{thres}} = 0.25$	70.61	33.08	72.45	74.07	78.16	10	55	4.8 (GPU)
		IoU $_{\text{thres}} = 0.5$	69.14	32.81	72.22	73.53	78.51	10	64	
		IoU $_{\text{thres}} = 0.7$	63.91	24.91	67.32	51.91	80.71	24	141	
FANTrack (IV'20)	2D + 3D	IoU $_{\text{thres}} = 0.25$	82.97	40.03	75.01	74.30	75.24	35	202	25.0 (GPU)
		IoU $_{\text{thres}} = 0.5$	80.14	38.16	73.62	72.71	74.91	36	211	
		IoU $_{\text{thres}} = 0.7$	62.72	24.71	66.06	49.19	79.01	38	406	
Ours	3D	IoU $_{\text{thres}} = 0.25$	93.28	45.43	77.41	86.24	78.43	0	15	207.4 (CPU)
		IoU $_{\text{thres}} = 0.5$	90.38	42.79	75.65	84.02	78.97	0	51	
		IoU $_{\text{thres}} = 0.7$	69.81	27.26	67.00	57.06	82.43	0	157	

Table 3: AB3DMOT results on the KITTI validation set [1] for “car” class, extracted from [8].

Category	Matching criteria	sAMOTA \uparrow	AMOTA \uparrow	AMOTP \uparrow	MOTA \uparrow
Pedestrian	IoU $_{\text{thres}} = 0.25$	75.85	31.04	55.53	70.90
	IoU $_{\text{thres}} = 0.5$	70.95	27.31	52.45	65.06
Cyclist	IoU $_{\text{thres}} = 0.25$	91.36	44.34	79.18	84.87
	IoU $_{\text{thres}} = 0.5$	89.27	42.39	77.56	79.82

Table 4: AB3DMOT results on the KITTI validation set [1] for “pedestrian” and “cyclist” classes, extracted from [8].

Method	Matching criteria	sAMOTA \uparrow	AMOTA \uparrow	AMOTP \uparrow	MOTA \uparrow
FANTrack	Dist $_{\text{thres}} = 2$	19.64	2.36	22.92	18.60
mmMOT	Dist $_{\text{thres}} = 2$	23.93	2.11	21.28	19.82
Ours	Dist $_{\text{thres}} = 2$	39.90	8.94	29.67	31.40

Table 5: AB3DMOT results on the nuScenes validation set [16] over all classes, extracted from [8].

Figure 13 shows that this system is capable of tracking objects even during long periods of time and with poor lighting conditions, as presented on the frame's right side on frames 13 and 28.

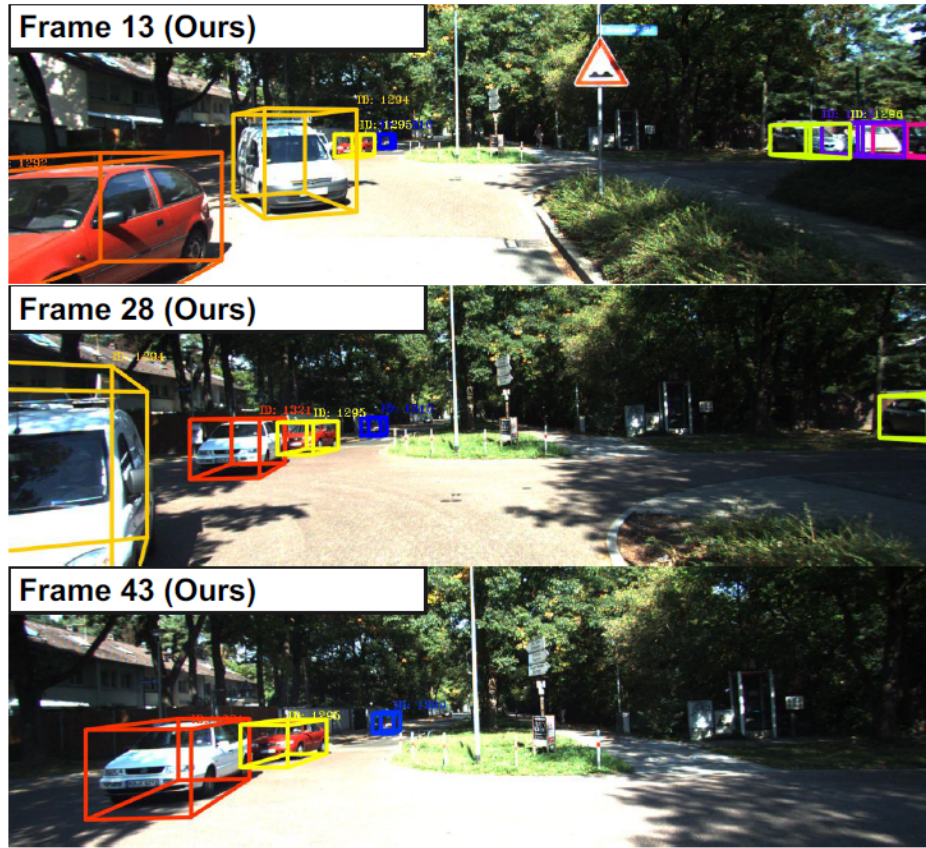


Figure 13: ABD3DMOT example, extracted from [8]

2.4 PROBABILISTIC 3D MULTI-MODAL

In 2020, [Chiu et al. \[9\]](#) approached MOT from a DL perspective, using DL models for the following modules:

- *Track initialisation*: while [Weng and Kitani](#) approached deterministically with a predefined number of frames before starting/ending a track, here it was treated as a binary classification problem. This model outputs a confidence score, and, for values above 0.5, a new track ID is initialised;
- *Fusion of 2D and 3D features*: each detection generates a vector of features, using its image and 3D detections. First, 3D coordinates are mapped into 2D, creating a map of features. However, instead of each detection containing only one vector, it contains its neighbours too, resulting in a 3×3 map (instead of just one value). Subsequently, the 3D bounding box is projected to the image to extract its information. Finally, all information is combined to return a complete feature map (3×3) for each detection;
- *Distance combination module*: its goal is to combine Mahalanobis distance [41] (i.e., a statistics method) and the deep feature distance. The latter combines the fused features from N detections and M tracks to

output a $N \times M$ feature map. In the end, both distances are combined to conduct data association using a greedy matching.

In terms of results, this research achieved better results (for AMOTA) than all compared MOT models that use the same object detector, in both KITTI and nuScenes datasets, in almost all categories. Even though they provide a LiDAR based MOT algorithm, its RGB and LiDAR version has better overall results on the nuScenes dataset, as presented in Table 7 and Table 8. “Pedestrian” is the only class that achieves worse results than previous models, most likely due to the changes that occurred over time regarding their appearance and geometry, as presented in Table 6.

Tracking method	Modalities	Input detection	Overall	bicycle	bus	car	motorcycle	pedestrian	trailer	truck
AB3DMOT	3D	MEGVII	17.9	0.9	48.9	36.0	5.1	9.1	11.1	14.2
ProbabilisticTracking	3D	MEGVII	56.1	27.2	74.1	73.5	50.6	75.5	33.7	58.0
CenterPoint	3D	CenterPoint	65.9	43.7	80.2	84.2	59.2	77.3	51.5	65.4
ProbabilisticTracking*	3D	CenterPoint	61.4	38.7	79.1	78.0	52.8	69.8	49.4	62.2
Our proposed method	3D	CenterPoint	67.7	47.0	81.9	84.2	66.8	75.2	53.5	65.4
Our proposed method	2D + 3D	CenterPoint	68.7	49.0	82.0	84.3	70.2	76.6	53.4	65.4

Table 6: Results on the nuScenes validation set [16] for multiple classes, extracted from [9].

Tracking method	Modalities	Overall	car
GNN3DMOT*	2D + 3D	29.84	-
PnPNet	2D + 3D	-	81.5
Our proposed method	2D + 3D	68.7	84.3

Table 7: Results on the KITTI validation set [1] for “car” class, extracted from [9].

Tracking method	Modalities	AMOTA	MOTA
GNN3DMOT*	2D + 3D	93.68	84.70
Our proposed method	2D + 3D	96.99	93.89

Table 8: Results on the nuScenes validation set [16] for “car” class, extracted from [9].

[9] used the KITTI 3D tracking dataset, splitting it into eleven validation sequences and ten training sequence, while the default split was used for the nuScenes dataset. Chiu et al. also discovered that their approach to the track life cycle brought significant improvements, as shown in ablation studies, resulting in less false positives.

As demonstrated in Figure 14, it rightly decides not to initialize new false positive tracks, since white bounding boxes appear in the left image, but not on the right, meaning that the track initialisation module decided, wisely, not to start tracks for the white bounding boxes.



Figure 14: Tracking example with wrong detections, extracted from [9].

2.5 PC-TCNN

In 2021, [Wu et al. \[10\]](#) did not focus on using tracking-by-detection, instead they focused on a joint detection and tracking approach, to explore the possibility of taking advantage of training the system as a whole, i.e., training detection and tracking together to improve the overall tracking performance.

This framework consists of three modules, as illustrated in [Figure 15](#):

- *Tracklet proposal generation*: unlike most approaches, the input is a sequence of N cloud point frames (last $N - 1$ frames and the current one). Afterwards, they are encoded into bird's-eye view feature maps (this representation is illustrated in [Figure 27](#)). Finally, it performs 3D object proposal generation and motion regression on these feature maps to generate a set of tracklet proposals;
- *Tracklet proposal refinement*: this module is responsible for capturing features from the spatial-temporal regions of interest on the point cloud sequence. Firstly, the points from each tracklet proposal are extracted by applying context-aware point cloud pooling [42], each containing local features. To explore local geometry, each point is then converted to bounding box coordinates, due to most objects sharing some consistent spatial-temporal features across consecutive frames, such as dimensions and volume. Then, this set of features is aggregated for each proposal. Finally, to refine each bounding box proposal during training, its characteristics, such as size, location and orientation, are regressed using its correspondent aggregated features. In the inference stage, NMS is used to obtain a set of refined tracklets;
- *Tracklet association*: at each frame, the refined tracklets are associated with previous trajectories using a greedy matching algorithm based on their 3D IoU. For successfully associated tracklets, the corresponding trajectories are updated with new info. Every unmatched tracklet will start new trajectories in the next frame.

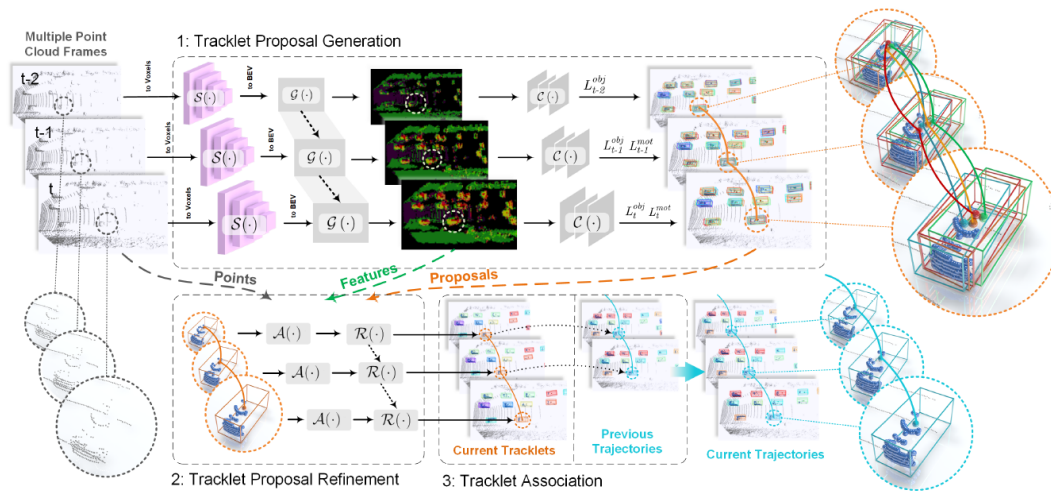


Figure 15: Tracklet Pipeline, extracted from [10].

The KITTI 3D Tracking dataset was used, focusing on IMU/GPS data, with the default split (21 training and 29 test sequences). Since the evaluation of the KITTI benchmark is done on 2D, tracking results were projected on a 2D image plane before the evaluation.

In terms of results, this paper achieved the best results when using 4 and 5 consecutive frames, for AP in 2D and 3D, respectively, as presented in Table 10. Overall and regarding MOTA, the best results are achieved with 4 consecutive frames as input. On the other hand, it is slightly over four times slower.

This method has the best MOTA score across all published research works. It also has a high recall score, as shown in Table 9, due to the exploitation of spatial-temporal features on point cloud sequences, improving the tracking accuracy of tracklets and reducing, e.g., false positives and ID switches.

Method	Input	MOTA	MOTP	Recall	Precision	MT	ML	IDS	FRAG
mono3DT	2D	84.52%	85.64%	88.81%	97.95%	73.38%	2.77%	377	847
TuSimple	2D	86.62%	83.97%	90.50%	97.99%	72.46%	6.77%	293	501
CenterTrack	2D	89.44%	85.05%	93.20%	97.73%	82.31%	2.31%	116	334
ODESA	2D	90.03%	84.32%	92.62%	98.77%	82.62%	2.31%	90	501
GNN3DMOT	2D+3D	82.24%	84.05%	-	-	64.92%	6.00%	142	416
aUToTrack	2D+3D	82.25%	80.52%	89.36%	97.03%	56.77%	7.38%	1025	1402
mmMOT	2D+3D	84.77%	85.21%	88.81%	97.93%	73.23%	2.77%	284	753
JRMOT	2D+3D	85.70%	85.48%	89.51%	97.81%	71.85%	4.00%	98	372
PointTrackNet	3D	68.24%	76.57%	83.56%	88.10%	60.62%	12.31%	111	725
ComplexerYOLO	3D	75.70%	78.46%	85.32%	95.18%	58.00%	5.08%	1186	2092
AB3DMOT	3D	83.84%	85.24%	88.32%	96.98%	66.92%	11.38%	9	224
PC-TCNN (Ours)	3D	91.75%	86.17%	96.08%	96.45%	87.54%	2.92%	26	118

Table 9: PC-TCNN results on the KITTI benchmark, extracted from [10].

Frames	AP@ <i>t</i>		MOTA	MOTP	Recall	MOT Metrics			IDS	FRAG	Speed (ms)
	2D	3D				Precision	MT	ML			
1	91.25%	91.16%	85.45%	86.23%	88.58%	97.17%	79.04%	1.81%	48	242	75
2	92.78%	93.12%	87.92%	86.14%	91.84%	97.10%	85.71%	2.29%	18	147	158
3	93.59%	94.18%	89.15%	86.19%	92.36%	97.95%	86.67%	1.80%	7	63	240
4	94.03%	94.48%	89.44%	86.10%	92.88%	98.02%	88.10%	1.80%	2	33	312
5	93.51%	94.49%	88.62%	86.16%	92.93%	96.67%	87.14%	2.29%	3	31	388

Table 10: PC-TCNN results on validation set with different number of point cloud frames, extracted from [10].

2.6 DISCUSSION AND ANALYSIS

In this section, state-of-the-art algorithms were presented, starting with two of the first 3D MOT, [6] and [7], given that these are much simpler than current approaches. Then, [8] provided a turning point in 3D MOT due to the metrics introduced by the research group, and the fact that the final algorithm is still a proper baseline for comparing with new approaches. It is also a good starting point to expand the proposed pipeline, because of its speed and simplicity across all modules. Furthermore, [9] explores the application of DL to different modules instead of a deterministic approach, such as track initialisation, proving that DL can be applied across all tasks in MOT. Finally, [10] is the top performing algorithm in the KITTI benchmark [1], which exploits the use of spatial-temporal features to improve bounding box proposal and tracking association.

The results achieved by all algorithms are shown in Table 11 and Table 12 on the KITTI and nuScenes benchmarks, respectively. As it can be observed, PC-TCNN [10] presents the higher performance among all compared algorithms by a significant margin, nonetheless, it only runs at 3 FPS on GPU, meaning that this approach is far from being capable of being applied in the context of AD. On the other hand, AB3DMOT [8] has the potential to be applied in AD due to its speed and low computational power needs despite its performance being far lower when compared to PC-TCNN [10]. Yet, due to its modularity and simplicity, this algorithm has room for improvement. BeyondPixel [6] and FANTrack [7] do not have many points in their favour.

Prob 3D [9] is harder to compare against others solutions due to the little information provided about all analysed algorithms in nuScenes benchmark, however it has a significant better performance than AB3DMOT [8], indicating that this algorithm is capable, in theory, of achieving great results on KITTI benchmark.

Algorithm	Data	HOTA \uparrow	FPS	Environment
PC-TCNN'21 [10]	LiDAR	81.1	3	GPU
AB3DMOT'20 [8]	LiDAR	69.8	207	1 CPU core
BeyondPixel'18 [6]	RGB + LiDAR	63.7	3	1 CPU core
FANTrack'19 [7]	RGB + LiDAR	60.8	25	8 CPU core

Table 11: Results of analysed architectures on KITTI benchmark [1], \uparrow denotes better.

Algorithm	Data	AMOTA \uparrow
Prob 3D'21 [9]	LiDAR	67.7
AB3DMOT'20 [8]	LiDAR	15.1

Table 12: Results of analysed architectures on nuScenes benchmark [16], \uparrow denotes better.

OBJECTIVES

The general purpose of **MOT** algorithms is to be able to follow the changing state of a target or multiple targets over time. They improve localisation estimation, reduce false alarms, deduce absolute velocity and trajectory, and generate a perception of the host's vehicle surroundings, where, typically, the states of the moving objects are focused on position, direction of the movement, velocity, and angular velocity. Under this project, an assessment will be made to tackle the tracking of multiple moving objects (e.g., pedestrians, cars, trucks, cyclists, etc.) in dynamic environments extended to the **LiDAR** 3D point cloud for **AD**.

3.1 GOALS

This dissertation's main goals are the following:

1. Research and develop multiple state-of-the-art approaches to track moving objects in the context of **AD**;
2. Benchmarking the performance and computational cost of the proposed solutions;
3. Identification of its main positive and negative aspects.

The first phase of this dissertation is more theory-based focusing on the in-depth analysis of 3D **MOT** systems in the context of **AD**. Firstly, it is essential to understand some concepts, such as **MOT** and the all involving components, evaluation metrics applied in this context, point clouds, their different representations and implications. At last, a study of state-of-the-art solutions is a relevant task for this dissertation in order to understand and analyse recent and successful approaches to tackle **MOT**.

The second phase is more development-based, where an initial temporal-consistent **MOT** will be developed and implemented. After that, it is necessary to optimise the overall performance of both object detector and tracker to better fit the temporal consistency in the context of **AD**. In the end, the results of the developed algorithm will be compared through a benchmark against other temporal approaches.

3.2 WORK PLAN

To better organise the work at hand, the project's timeline was divided into easier and shorter tasks, presented in Table 13, all leading to this dissertation's target, ending with the following work plan:

Scheduling:**First phase – Pre-dissertation:**

1. Literature review of methodologies that fall under the same topic of research;
2. Definition of problem restrictions;
3. State-of-the-art and research plan elaboration;
4. Analysis of the current 3D object detection and tracking algorithms in the context of [AV](#).

Second phase – Dissertation:

1. Propose a solution of a consistent temporal object tracking;
2. Optimize current 3D object detection algorithm to better fit the temporal consistency;
3. Iterate and evaluate the obtained results through benchmarks when applied to [AD](#).

	2021			2022								
	Oct.	Nov.	Dec.	Jan.	Feb.	Mar.	Apr.	May	Jun.	Jul.	Aug.	Sep.
Literature review												
Problem restrictions definition												
State-of-the-art research												
MOT algorithms analysis												
MOT solution proposal												
Object detector optimisation												
Report results												

Table 13: Work plan schedule.

3.3 CONTRIBUTIONS

The main contributions of the present dissertation are:

- Explore the viability of LiDAR sensors to improve the performance of 3D tracking systems;
- Analysis of state-of-the-art solutions for AD 3D tracking systems;
- Analysis of datasets' characteristics for the problem at hand;
- Research and implement a tracking solution that takes advantage of DL's flexibility to improve its performance. This architecture was divided into separate modules and uses detections produced from a 3D object detector. Additionally, two DL models were implemented, responsible for state prediction and association across detections and stored states, two of the most crucial tasks in tracking systems. Furthermore, new components were also developed to mitigate common problems, such as detection uncertainty and overlapping prediction. A new feature was implemented into the tracking architecture, inspired by a typical detection feature, and is responsible for selecting/removing overlapping objects. This proposal was trained and tested using a public dataset widely studied in state-of-the-art methodologies;
- Optimise said solution with a hyper-parameter search to reach the highest performance possible;
- Benchmark developed solution with state-of-the-art results, as well as, verifying main advantages and limitations;
- Analyse the feasibility of this implementation in the context of AD, regarding its performance and inference speed, while considering resources needs to run the algorithm on an embedded platform.

Part II

CORE OF THE DISSERTATION

LIDAR BASED OBJECT TRACKING

Object tracking is used for multiple applications and involves different types of input data. For example, a [MOT](#) system refers to an algorithm that receives, usually for every frame, a set of object detection outputs and assigns to each of them a unique ID. In short, the main challenge associated with object tracking is achieving an *equilibrium* between computational efficiency and performance.

In this section, some available datasets will be presented, along with its possible data representation formats, the proposed pipeline along with the expected output and the considered evaluation metrics.

4.1 DATASETS

During the research for the state-of-the-art, some datasets stood out for the frequency with which they appeared. Typically, all projects used either the KITTI or nuScenes datasets, often both. Therefore, the three most used datasets were selected for benchmarking. In addition, KITTI-360 was also included. To allow for a better and more informed choice, the values/information were gathered in [Table 14](#).

[Table 14](#) compiles characteristics from the mentioned datasets. Its purpose is to support and justify the dataset selection, by comparing the datasets' characteristics. The selected features are:

- **Frames-per-second (FPS):** A object tracker uses multiple frames and, therefore, more frames results in more information which is good for the algorithm, and allows the algorithm to be more responsive and have newer information;
- **Number of classes:** More classes allow for a more detailed output, however, too many possible classifications during detection can “confuse” the algorithm;
- **Density (Number of points):** More [LiDAR](#) points during detection, usually, result in better detection due to its better portraying its environment;
- **Range:** Higher range allows to collect information from farther objects which will allow to better anticipate future events;

- **Sequences and Frames:** Usually autonomous driving datasets account from enormous amounts of data, and it is not possible to collect it all continuously. Therefore, the data is divided into sequences that represent different scenarios in order to have different tests to prepared for the algorithms. Each sequence is divided into frames, and these two are important factors in dataset selection. More sequences and/or frames imply more diverse data and/or more quantity of data;
- **Frames per sequence:** Object tracking algorithms usually are more prone to mistakes in the beginning of sequences, because all objects are new and it has less confidence in them. Thus, more frames for each sequence allow the algorithm to build up its confidence, stabilise its performance;
- **Number of 3D bounding boxes:** An object tracker tries to learn how to predict and track objects through bounding boxes. Therefore, more bounding boxes increase the amount data to learn patterns, improving its performance;
- **Driving scenarios, Weather and Time of day:** These three factors are important to test and train the algorithms in different scenarios.

Dataset	KITTI [1]	KITTI-360 [43]	nuScenes [16]	Waymo [44]
FPS (LiDAR)	10	10	2	10
# Classes	8	19	32	4
Density (# Points)	100k	100k	35k	177k
Range	80-120m	80-120m	70m	short/mid-range
# Sequences	50	11	1000	1200
# Frames p/Seq	≈ 380	n/a	40	200
# Frames	≈19k	n/a	40k	240k
# 3D Bbox	200k	68k	1.2M	12.6M
Driving scenarios	rural, urban areas, highway	suburban areas	urban areas	urban areas
Weather	dry	n/a	dry, rain	dry, rain
Time of day	day	n/a	day, night	day, night

Table 14: Dataset features comparison

By analysing Table 14, we can see that nuScenes stands out from others due to its refreshing rate (FPS), point density, number of sequences, and frames. These first two points highlight this dataset's main weaknesses. Given the project at hand, point density is an important feature, allowing a better comprehension of the scene layout. On the other hand, it has a much higher number of sequences and frames, however its sequence duration is much shorter (around 40 frames). Furthermore, nuScenes has a higher number of classes yet most of them are unnecessary for this use case, e.g., classes "animal" or "sidewalk". Regarding the sensors' range, all have somewhat the same reach.

Waymo was dismissed due to incomplete information regarding its characteristics, documentation and lack of available evaluation tools.

KITTI has less diversity in terms of weather and time of day, as its data was collected throughout the day (only in dry weather), however it is more diverse regarding the types of roads where data was recorded, i.e., rural, urban, and highway scenarios, as opposed to urban and suburban areas in the other datasets. KITTI's main advantages are: (i) the fact that it is the most used dataset for tracking benchmarking; (ii) the available support and documentation on the internet. On top of that, KITTI also has a longer average duration per sequence, being almost twice as long as the second longest average sequence, Waymo [44]. Therefore, KITTI and KITTI-360 appear to be the better choices. Regardless, further investigation on Waymo's possible qualities and positive points will be conducted, which may be used for evaluation purposes. An example of an annotated 3D point cloud from KITTI dataset is presented in Figure 16.



Figure 16: Example of KITTI dataset annotated 3D point cloud, obtained by [11]

4.2 DATASET ANALYSIS

In this section, a brief analysis for KITTI dataset [1] will be presented, focusing in its interesting features, such as number of objects, number of objects per frame, among others. Additionally, [1]'s recording platform setup is available on its [website](https://www.cvlibs.net/datasets/kitti/setup.php)¹, as well as the specifications of its sensors, which are:

- 1 inertial navigation system (GPS/IMU), OXTS RT 3003, used to record ego-motion;
- 1 laser scanner, Velodyne HDL-64E, in this case, a LiDAR sensor;

¹ Available at: <https://www.cvlibs.net/datasets/kitti/setup.php>

- 2 grayscale cameras with 1.4 Megapixels, Point Grey Flea 2 (FL2-14S3M-C), mounted approximately level with the ground plane;;
- 2 colour cameras with 1.4 Megapixels, Point Grey Flea 2 (FL2-14S3C-C), mounted approximately level with the ground plane;;
- 4 varifocal lenses, 4-8 mm: Edmund Optics NT59-917,

Regarding data acquisition, it uses the following process. Both the laser scanner and the cameras work at 10 FPS. For each scanner's spin, cameras take 1 picture triggered when the scanner is facing forward. Figure 17 shows the sensor setup and an image of the fully equipped vehicle.

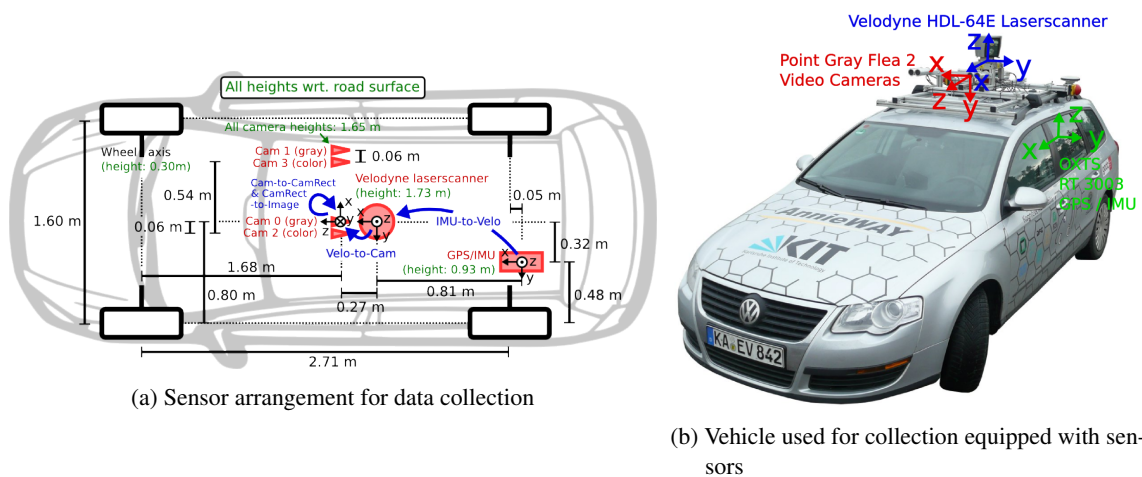


Figure 17: Platform used for data acquisition, extracted from [12]

Usually, a dataset is divided into two or even three splits: training, validation and testing. [1] already has two divisions, training and testing, however, the testing split does not contain any ground-truth, which would not allow to benchmark the algorithm's performance, since we do not have the correct output. Therefore, it was decided that the training split would be divided into two parts to be used in two tasks: training the learning model and validating its results.

Additionally, in order to have comparable results to the state-of-the-art, it was used the same split as [8] and [10]. The data split considered uses sequences 1, 6, 8, 10, 12, 13, 14, 15, 16, 18, 19 for validation, while the other 10 sequences (0, 2, 3, 4, 5, 7, 9, 11, 17, 20) are used to train the learning modules.

The dataset encompasses 8 classes:

- Car, as the name suggests represents a car;
- Pedestrian represents any person on the road or sidewalk;
- Cyclist corresponds to a person riding, necessarily, a bicycle - i.e., motorcycle are excluded;

- Van correspond to a van vehicle;
- Truck represents a truck vehicle;
- Person represents a person sitting;
- Tram represents a Tram;
- Misc is reserved for every object that was annotated that does not fit in any category.

Figures 18 and 19 illustrate the distribution of the classes of all annotated objects in total values and percentages, respectively. Note that the scale of the first diagram is logarithmic to make it easier to perceive the differences between the lower values.

It is easily observed that the Car class is, by far, the class with the higher number of annotations, with more than 27000, representing more than half of all annotations, while the second highest class, Pedestrian, has almost 11500 annotations. Overall, there are 47262 annotations in the dataset.

This discrepancy will most likely create a tendency for the tracking and detection algorithms to prioritise these classes' performance over the others. Furthermore, considering the class Van (third most represented) is accepted in the Car class due to the classes' similarities, which means that the class Car has approximately 30600, representing more than 64% of all labels. Additionally, the class Person is also considered as Pedestrian, raising the number of annotations to more than 12000. These factors will make this representation difference even more pronounced, meaning, possibly, a lower performance of both detection and tracking.

Finally, after some consideration, it was decided to target 3 of these classes, Car, Pedestrian and Cyclist, since, usually, these are the evaluated classes for the detection and/or tracking tasks in this dataset. Nevertheless, 2 of these classes (Cyclist and Pedestrian) tend to be smaller objects making them harder to detect, so it is expected that the performance for these classes will be even lower.

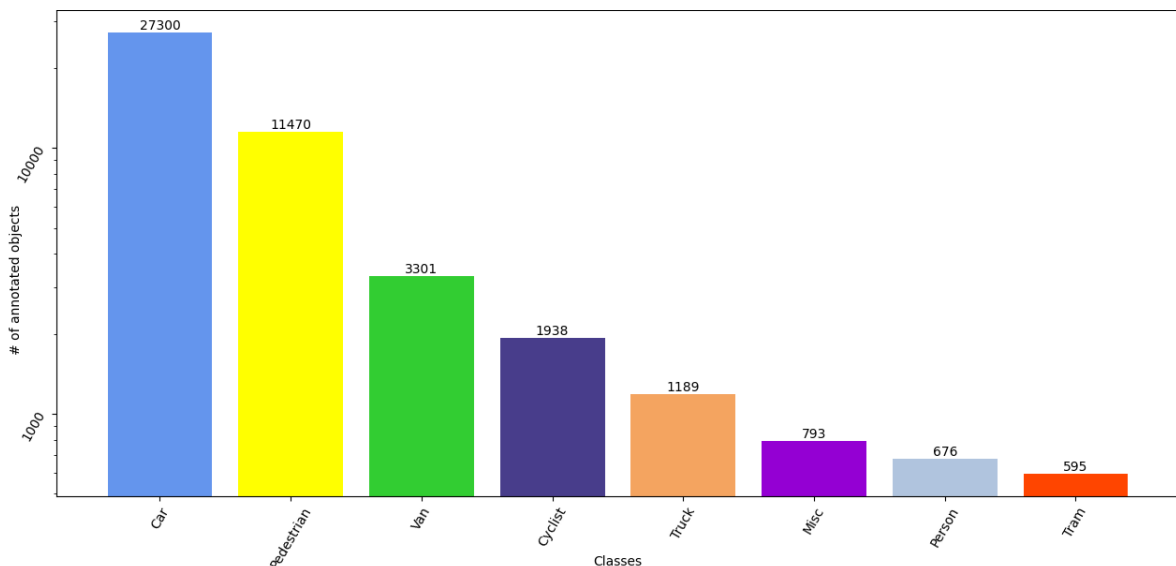


Figure 18: Number of objects per class

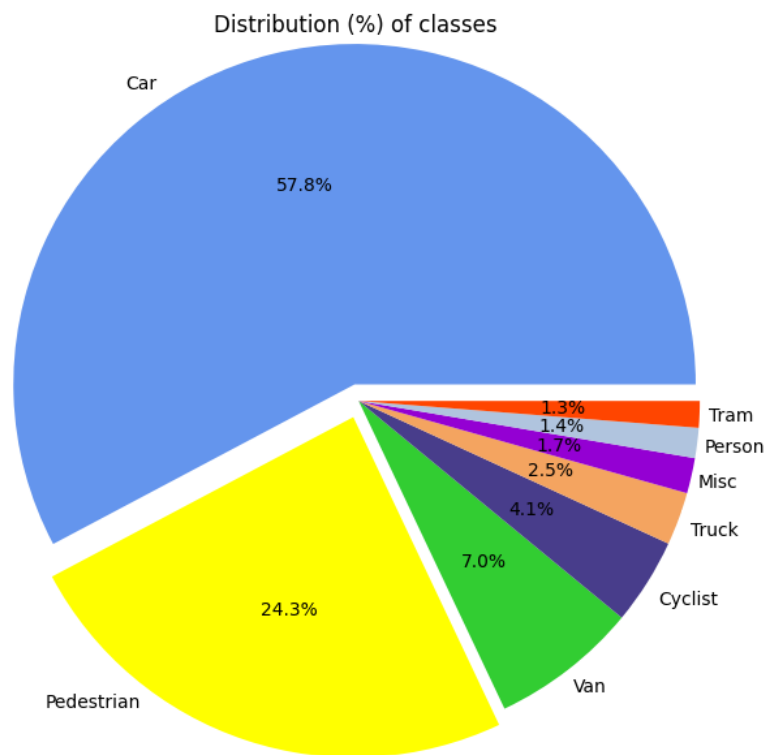


Figure 19: Classes distribution in percentage

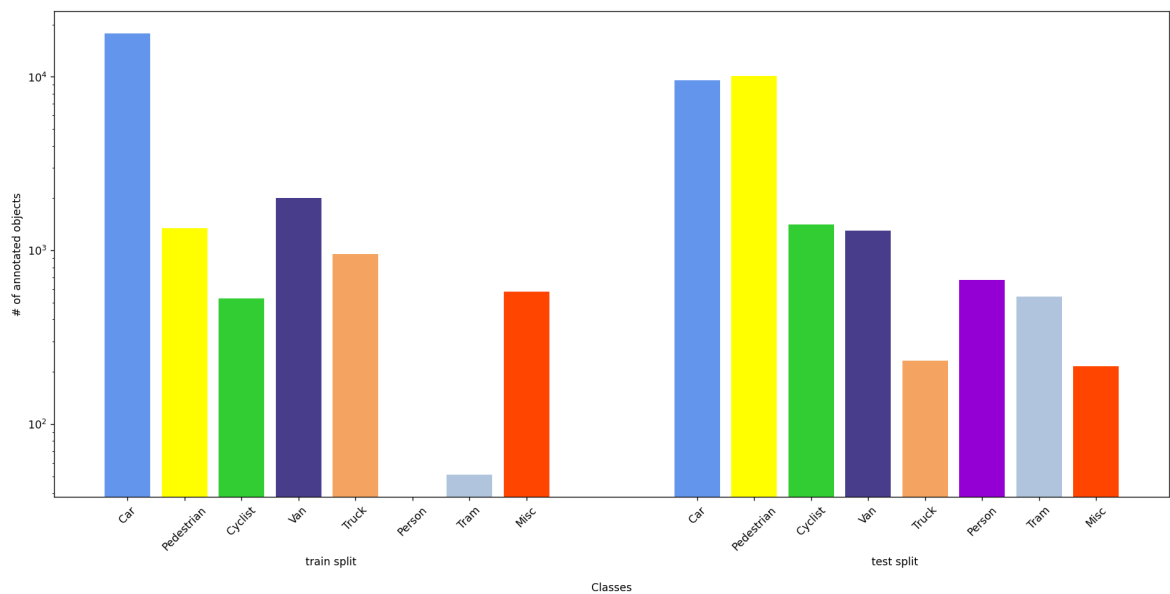


Figure 20: Number of objects per class for both splits

Due to dividing the dataset into two splits, classes' representation can become unbalanced. Thus, it is important to ensure the distribution remains similar for both splits. These are depicted in Figure 20. After analysing this plot,

some discrepancies appear: (i) the train split has a similar proportion between Car and Pedestrian objects to the overall's. In contrast, the test split has a almost the same number of Car and Pedestrian objects; (ii) Person is not represented in the train split; and (iii) Tram is also much less represented in train split.

However, if this analysis is restricted to only the selected classes, i.e., Car (including Van), Pedestrian (including Person), and Cyclist, the distribution of both splits would be quite similar to the overall distribution, thus, it can be concluded that the sequence distribution is not be detrimental to the algorithm's performance.

The KITTI Tracking [1] is, as the name suggests, a tracking dataset, and one of the most relevant characteristics is the objects' lifespan, i.e., a time window (usually in frames) in which the objects exist and are labelled. This distribution can be seen in Figure 21, and, after a quick analysis, we conclude that most objects only exist for around 20-50 frames which corresponds to 2-5 seconds. The object with the longest lifespan has more than 600 frames or 60 seconds. Although, this situation is the exception to the rule as the majority of the object appear for less than 100 frames (10 seconds) as it can be observed in Figure 22. This is expected given that many objects are either (i) parked/stationary, (ii) vehicles in the opposite lane, (iii) objects that are not following the same route as the recording test vehicle, or (iv) objects with drastically different speeds, i.e., slower or faster than the recording test vehicle, resulting, eventually, in an overtake or be outpaced, respectively. All of these cases represent objects that will only appear for a short period of time.

Having access to higher object lifespans is important for tracking algorithms since it maintains the objects for longer, allowing for a more consistent tracking and it makes the algorithm less reliable on its input detections, as it stores more information about the objects in the environment.

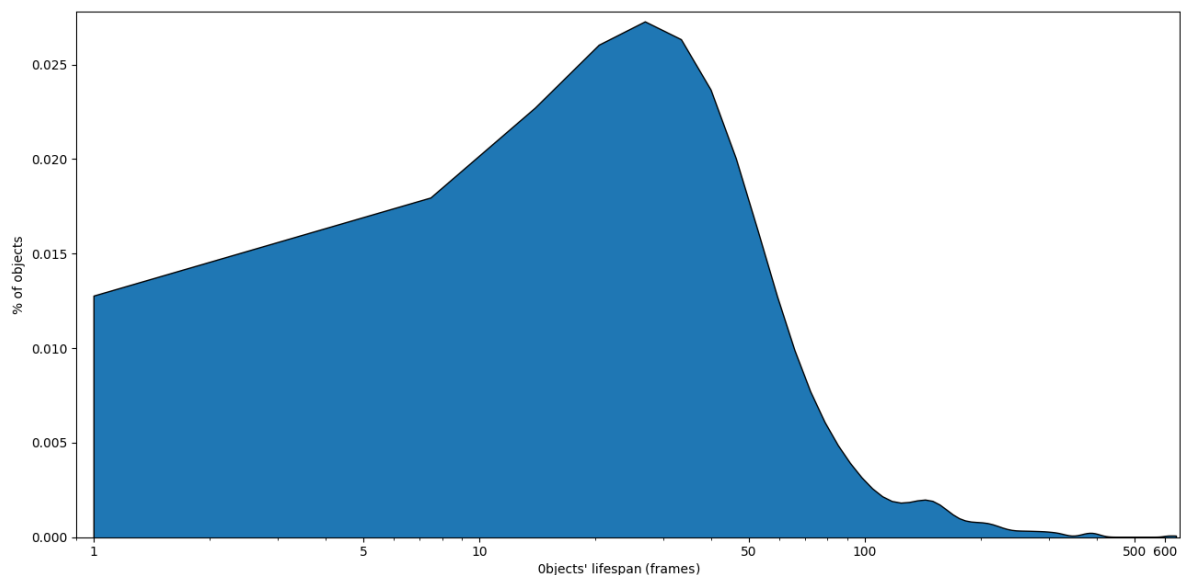


Figure 21: Distribution of objects' lifespan

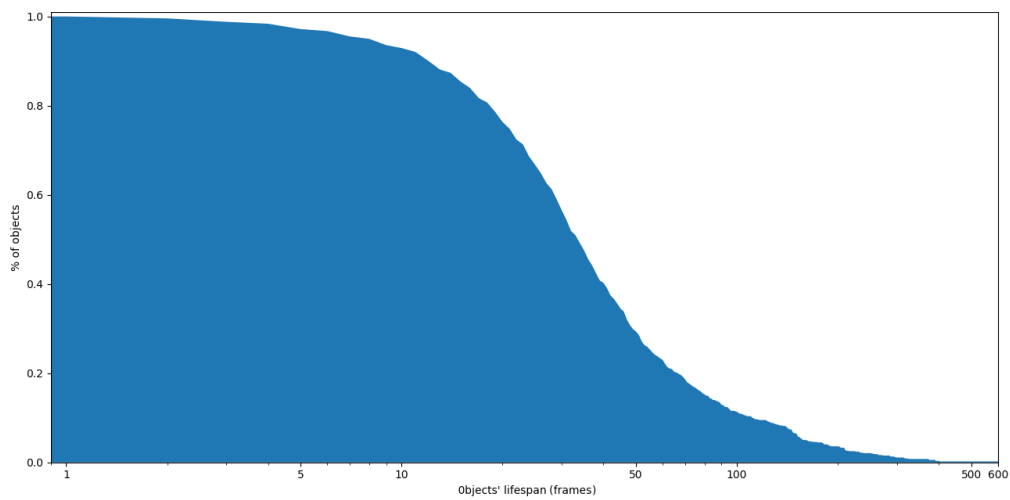


Figure 22: Cumulative distribution of objects' lifespan

Another relevant attribute for such algorithms is working properly in clustered environments, and in Figure 23 it is shown that most frames have a reasonable number of objects per frame, 1 to 7 objects, while 3 is the most common number of objects per frame. Even though, there is still a significant percentage of frames with 10 or more objects, around 20% as it can be observed in Figure 24. This value should be enough to allow the algorithm to learn how to behave in crowded environments. Nevertheless, ideally there should be more frames with a high number of objects to create harder “challenges” for the multi-tracking. It is also relevant that when referring to 10 objects, not every one of them is a vehicle, so the fact that most of the frames only contain a few objects indicates that the dataset was acquired in more quiet scenarios.

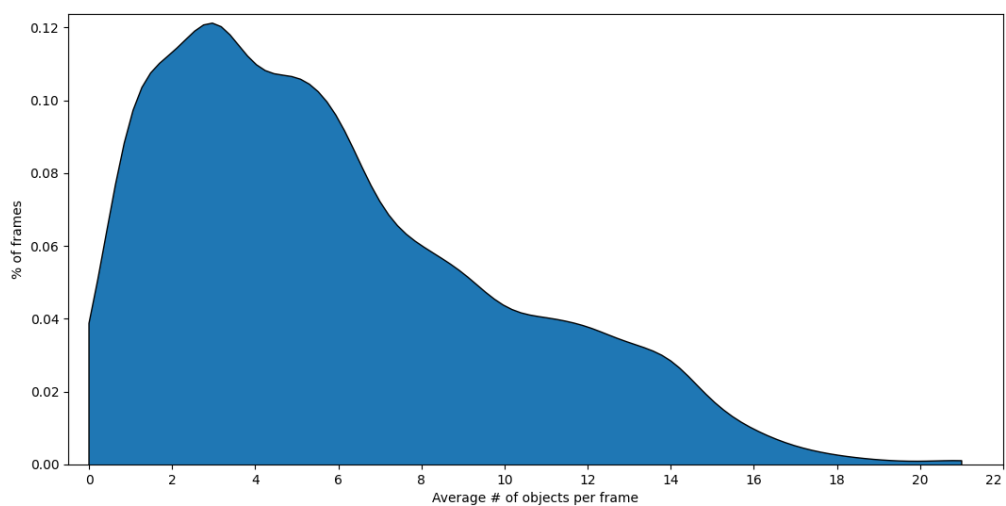


Figure 23: Distribution of number of objects per frame

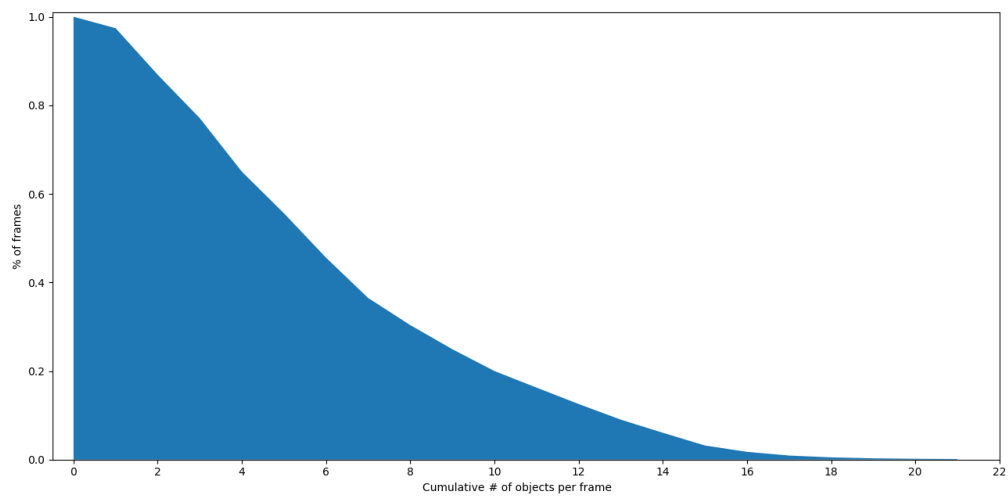


Figure 24: Cumulative distribution of number of objects per frame

Another interesting characteristic is the objects' relative positioning to the recording test vehicle. As displayed in Figure 25 most of the objects appear to be aside and less than 50 metres of distance. This variable is one of the most relevant to ensure good detections, since, as expected, the farther the object is, the smaller the amount of LiDAR points that will be received by the sensor, causing the object to be less represented in the search space. Most objects are to the right because the objects that will appear for longer continuous periods of time are the vehicles in front. And since this dataset was collected in Germany, where the vehicles drive on the right side, combined with the fact that the origin of the objects' relative position is the left camera (as demonstrated in Figure 17a), most of these objects will be slightly to the right, hence the tendency to have objects slightly to the right as represented in Figure 25.

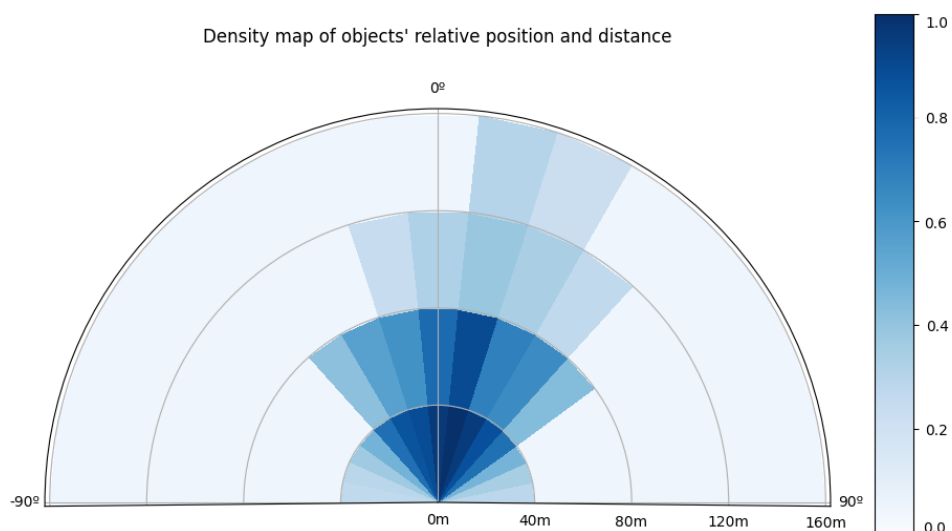


Figure 25: Relative distance and positioning density

Finally, Figure 26 depicts the distribution of objects' bounding box dimensions for each class, and as expected, Person, Pedestrian and Cyclist are, commonly, smaller objects. This corroborates with what was said before. These classes are indeed smaller and can constitute a performance issue in the future. Additionally, these dimensions will mainly impact the tracker's performance through the object detector as smaller objects are harder to detect.

Finally, this plot also allows us to foresee possible mistakes during detections. When looking at dimensions only, we notice that almost all classes have unique distributions except the pair Car-Van and Person-Pedestrian. These similarities help to justify the grouping of these classes.

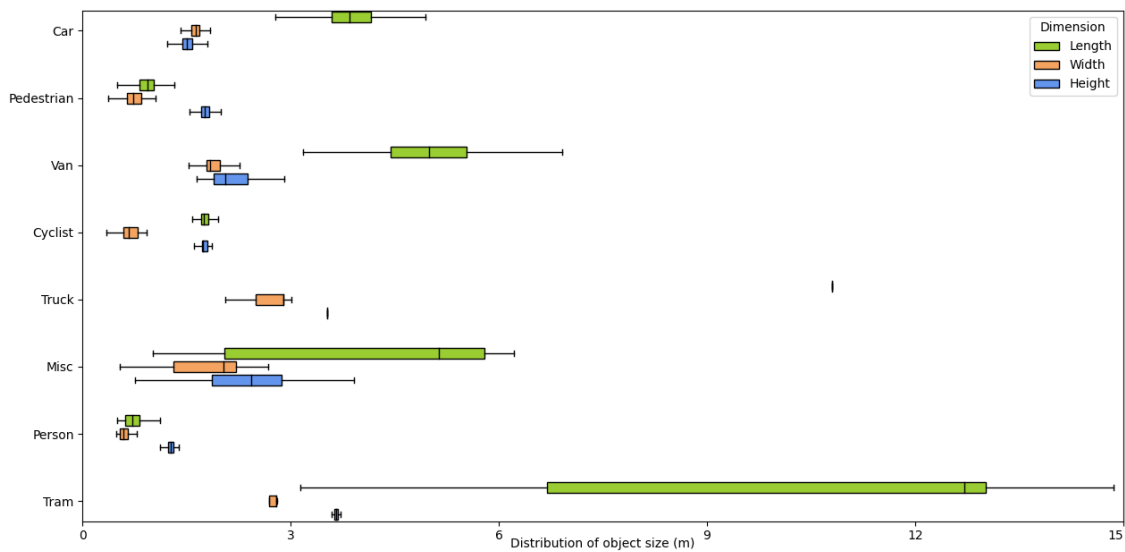


Figure 26: Objects dimensions (length, width and height) distribution

4.3 DATA REPRESENTATION

The 3D LiDAR datasets consist of point clouds (i.e., a set of unordered 3D points). This data is characterised by its sparseness, given that, the further away from the sensor, the lesser points are acquired, ending up with a less dense object representation for faraway objects. Another limitation is the occlusion that occurs when an object is between the sensor and other objects, resulting in empty zones in the point cloud.

Generally, even though its native representation is a sparse point cloud, there are many ways to represent them, as illustrated in Figure 27. These are described as followed:

- Raw points: most simple form of representation. No information is lost, i.e, each 3D point is kept, yet it does not exploit any geometric properties. Each point stores information about its reflectivity [13];
- Range view: is the equivalent of representing the point cloud as a 360° image resulting in a 2D representation. Essentially, the 2D axes correspond to the azimuth and elevation angle, while the value of each pixel

corresponds to the distance from the nearest point within the corresponding frustum, and, if no point is found inside, it is assigned with zero [13], as depicted in Figure 27b;

- 3D voxels: consists of dividing the 3D space into non-overlapping voxels, all with the same dimensions, resulting in a 3D grid. Each voxel contains information about the points inside it, e.g., a binary value to indicate whether or not it contains points, or the number of points. While it needs less memory, this representation does not consider specific properties of the point cloud, and most voxels are empty, resulting in a sparse representation [13], as depicted in Figure 27a;
- Bird's-Eye View (BEV): similar to 3D Voxels, nevertheless, while voxelization maps a 3D representation, here the height is omitted, and the height of the generated voxels is the height in 3D space [13], as depicted in Figure 27c.

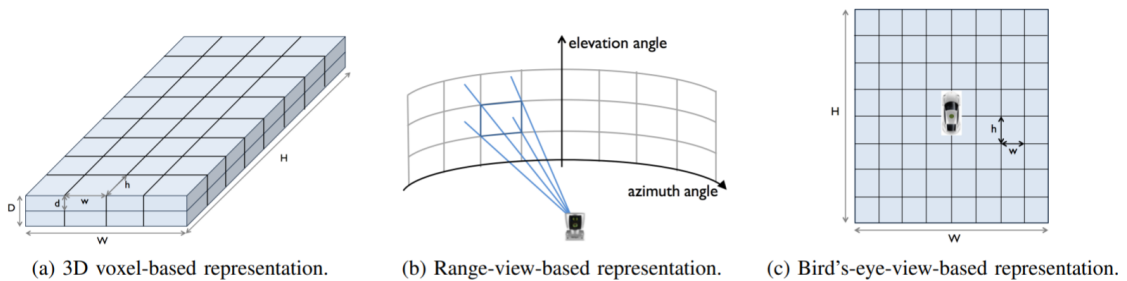


Figure 27: Illustration of most common techniques to encode 3D points, extracted from [13]

4.4 DETECTION MODEL

In order to develop a object tracking algorithm, the first step was to select and integrate an object detection model. After analysing the KITTI detection benchmark, available in [17], the clear choice was the SE-SSD [14] algorithm, given that it was one of highest ranked algorithms that used only LiDAR information and its code repositories² were available to train and test this algorithm, while having relatively fast inferences times.

SE-SSD's architecture uses 2 single-stage detection (SSD) networks, i.e., the classification and detection of objects are performed jointly instead of detecting objects and then classifying them. These 2 networks have different training processes: one is named the teacher and uses little to no data augmentation on its training data, while the other is called the student, whose training data is heavily modified by data augmentation. The techniques used are: (i) mix-up, i.e., introduce data from different scenes; (ii) random rotation and translation for individual points; and (iii) random rotation, translation and flipping for whole scenes. This detector applies the following pre-processing method: convert the sparse information into 3D voxels and a BEV representation. An overall representation of [14] architecture is depicted in Figure 28.

² Available at: <https://github.com/Vegeta2020/SE-SSD>

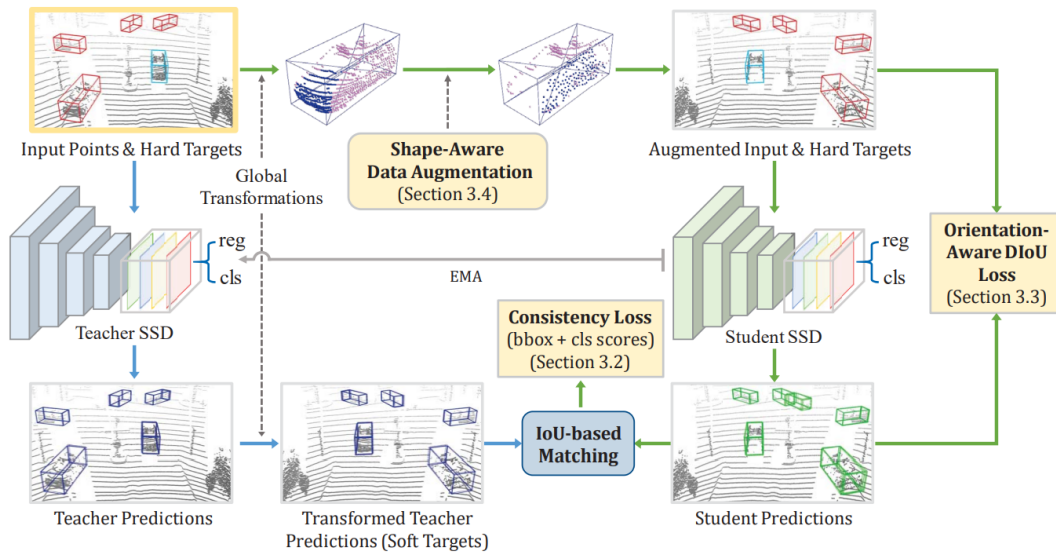


Figure 28: SE-SSD [14] architecture

However, after a few modifications to train and predict for multiple classes, its executing time dramatically increased, needing almost 60 days to fully train and 6 hours to perform predictions for every frame (this task would take only 3 and a half minutes before when it was only prepared for a single class).

Initially, SE-SSD was chosen because it ranked 3rd (when comparing AP for medium difficulty) for LiDAR only detector and it was roughly 3 times faster than the first two (around 0.08s per frame), while using similar resources. After discarding this model for not infer for all desired classes, the best available model that fitted the criteria was PV-RCNN. Its downside is the loss of performance for the class Car and the increase in execution time, however this was already expected since it is performing detections for 3 classes ((i) Car, including Van; (ii) Pedestrian, including Person; and, (iii) Cyclist). The performances each class from both algorithms are described in Table 15.

Algorithm	Car	Pedestrian	Cyclist	Runtime (ms per frame)
SE-SSD [14]	82.54%	—	—	30ms
PV-RCNN [15]	81.43%	43.29%	63.71%	80ms

Table 15: Comparison between implemented detection algorithms, AP values for medium difficulty, according to [17]

Both algorithms have equal outputs, each detection is characterised by its object class, its occlusion and truncation (both values are usually set to 0), its rotation in relation to the camera, its 2D image coordinates, its dimensions, location and orientation, and, finally, the models' confidence for said detection.

PV-RCNN uses a two-stage detection approach, taking advantage of both point-based, voxel-based, and BEV-based features. PV-RCNN's architecture is characterised by its PointNet-based network and 3D Voxel Region-based CNN, hence the name PV-RCNN. However, this work has some innovations. For instance, it uses multiple

voxel features obtained at different scales instead of only one grid to perform detections. This algorithm uses four distinct feature maps with different voxelization scales (including a **BEV** feature map). In terms of data augmentation, it employs random flipping, scaling and rotation, and mix-up. An overall representation of PV-RCNN architecture is depicted in Figure 29.

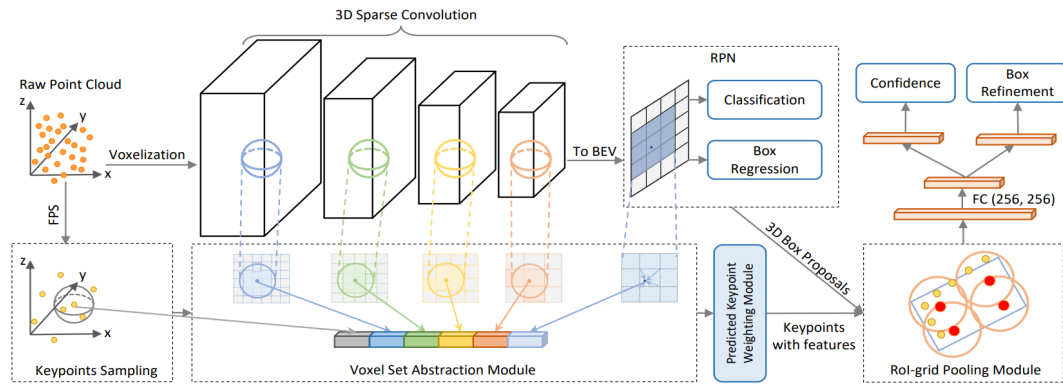


Figure 29: PV-RCNN [15] architecture

Additionally, PV-RCNN [15] allowed to take advantage of Transfer Learning (TL), i.e., using pre-trained weights of the architecture trained on similar object detection datasets. And, since tracking is this work's main focus, and [15] already had acceptable results no further improvements were made regarding detection. These weights are available on the [open-mmlab GitHub repository](https://github.com/open-mmlab/OpenPCDet/)³.

4.5 PRE-PROCESSING

Given the proposed pipeline, the tracking algorithm receives detections from an independent module. For this, all detection were done previously and stored in files, enabling reproducibility. However, KITTI object detection dataset (dataset to which detection algorithms are trained and validated) structure is slightly different from the object tracking algorithms. Therefore, the first step was converting the desired dataset [1] to follow a standard structure to be worked on, as illustrated in Figure 30.

³ Available at: <https://github.com/open-mmlab/OpenPCDet/>

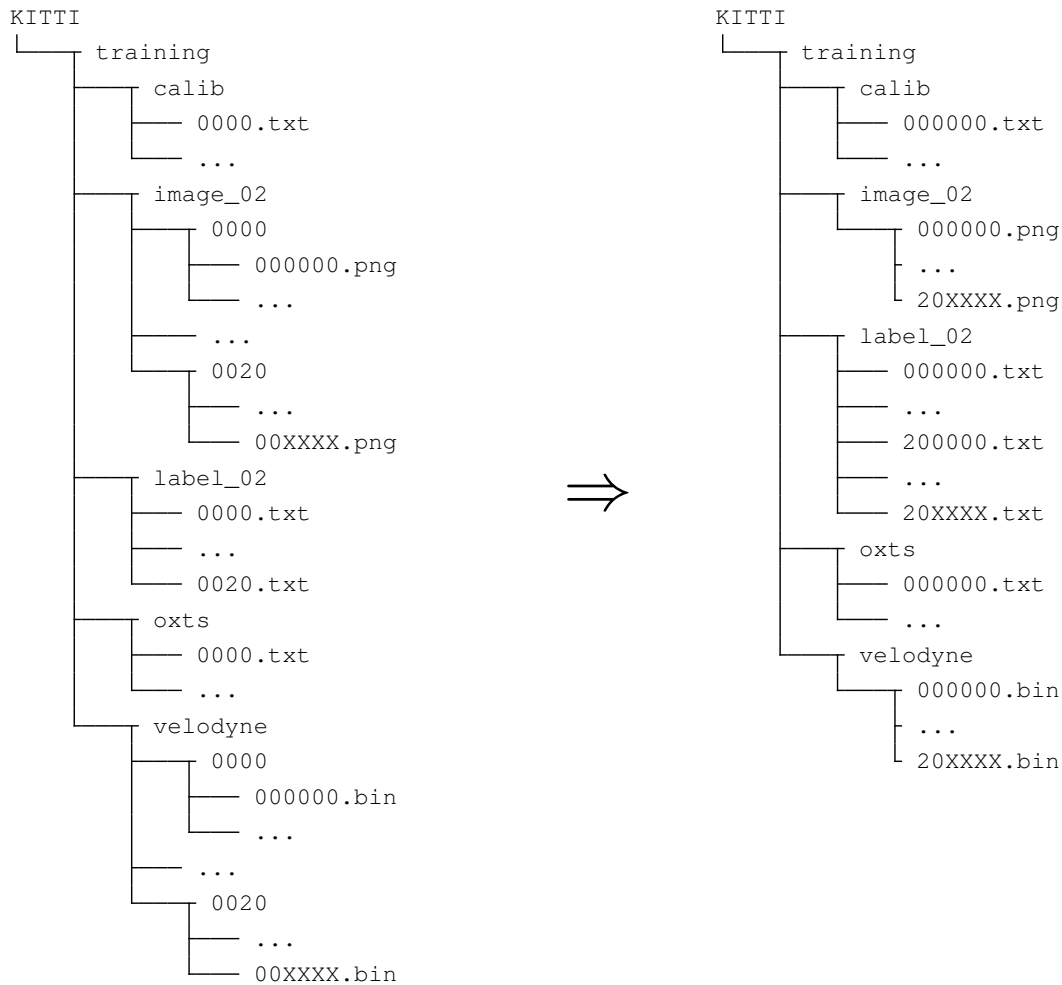


Figure 30: Folder structure adaptations to comply with object detections' structure

After completed this gathering and dataset preparation task, it must be processed before being fed into the architecture. The first attempt was to use the detections made by the object detection algorithm to train multiple modules, however, given that there is no information available to associate objects across different frames, this approach was discarded. After this failed attempt, it switched to the dataset labelling, where there is information to match the same object through different frames. Using this approach, it was trivial to use N consecutive states to then predict the next one.

After deciding on the division of sequences and which data will be used, it is fundamental to extract the desired information, i.e., prediction and association.

Regarding the prediction of future states, it was used a sliding window to create the necessary train use cases, i.e., use the states of a given object since frames i until $i + n - 1$ to predict the state of the next frame, frame $i + n$, as depicted in Figure 31. This information is then used as input, with the following format, $N \times M$, and its output is $1 \times M$, where M represents the number of variables that describe the object's state. However, these M output variables represent the difference from the last state (i.e., the velocity, both linear and angular) rather

than the actual state. This modification is due to result in better results than its alternative, which predict the state itself.

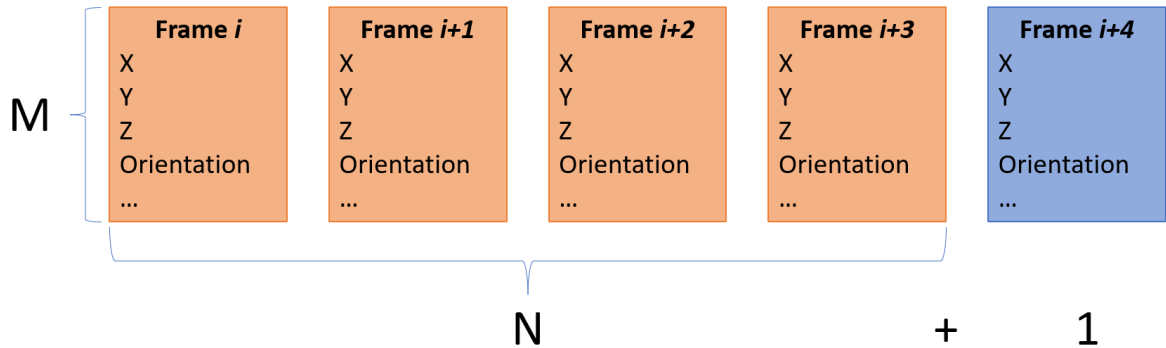


Figure 31: Prediction information representation of a single object, orange represents ground-truth input data, while blue represents the expected output

For association, K variables are used to describe the objects' states, and the input data is presented in the shape $2 \times K$, returning a single value, i.e., the probability of these two states corresponding to the same object, In a further iteration, the input data was changed to $1 \times K$, where it passed the variance between the two states, instead of the states themselves, as shown in Figure 32. Again, this modification resulted in improved results, most likely due to having values closer to 0, allowing the model to converge faster. Note that K differs from M since only variables regarding objects' position and angle are applied for object states' predictions, while for the object association, dimensions are relevant too.

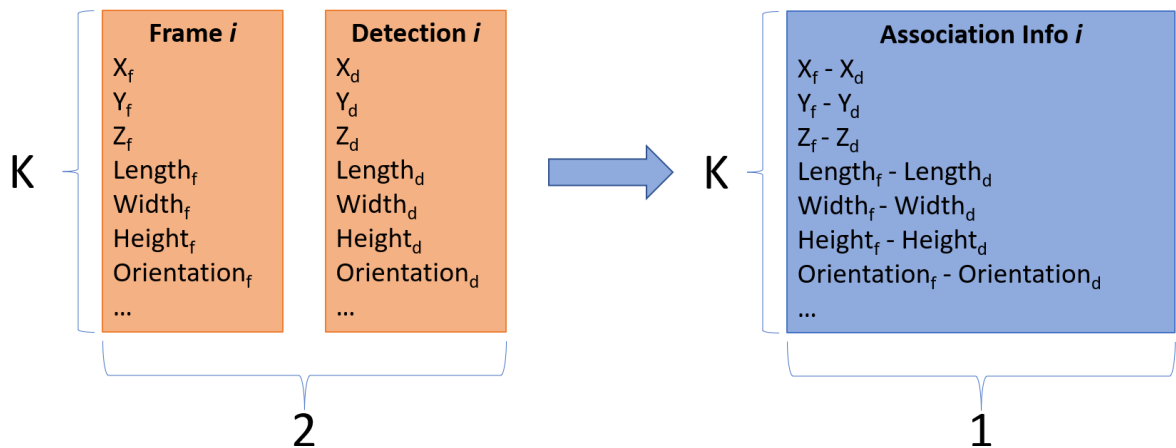


Figure 32: Object association information representation, orange represents the original approach, while blue represents the final, both approaches return a number (confidence) ranging from 0 to 1

4.6 DATA AUGMENTATION

Data augmentation is one of the most sought-after topics, with almost everyone using it for nearly all tasks, as it is one of the most relevant steps when developing deep learning models, since it improves models' consistency and performance. It creates new data out of the real data. The most common augmentation techniques are:

- **Translation:** Typically, it shifts an image resulting in a similar yet displaced image. In this case, it translates to moving the bounding boxes along their axis. Nevertheless, adding noise will already move the bounding boxes, so it was decided not to implement this technique as it would not bring benefits;
- **Rotation:** As the name suggests, in the case of images, it rotates by an arbitrary angle. This type of transformation is hard to implement due to having relative coordinates, so we can not ensure the same rotation is applied to all frames in a sequence. Therefore, this technique was not considered;
- **Scaling:** This technique scales up or down the image. In our case, it affects the dimensions of the objects bounding boxes. Though adding noise will alter their sizes. For this reason, it was decided not to implement this technique;
- **Flipping:** Commonly, flipping in computer vision refers to flipping an image either on the vertically, horizontal axis or both. In this case, a bounding box has its coordinates flipped, i.e., the x coordinate changes to $-x$;
- **Noise:** Usually, it refers to making images blurry. In the 3D context, we add noise to the state (position, size and orientation) which will result in similar states yet with small variations. It will encourage the algorithm to be more flexible and better at adapting to different situations.

We implemented also a new technique, inverting trajectories, where instead of training with a trajectories described from frame i through $i + 4$, $i + 4$ is its initial state, while i is the final step, i.e., reversing its path and velocity.

Figure 33 illustrates examples of data augmentation techniques for both images and bounding boxes.



(a) Sample image, extracted from [45]

(b) Translation example



(c) Rotation example



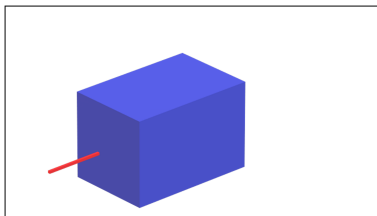
(d) Scaling example



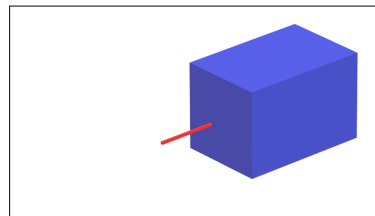
(e) Flipping example



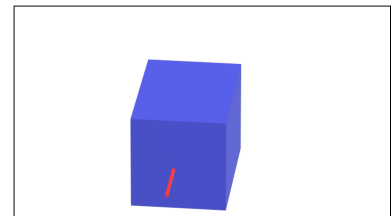
(f) Noise example



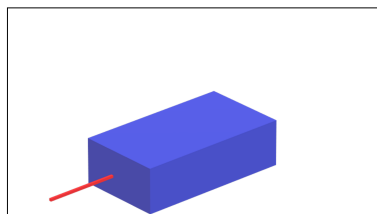
(g) Sample bounding box



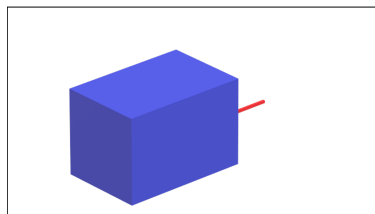
(h) Translation example



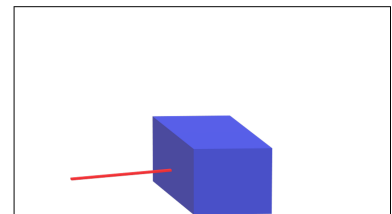
(i) Rotation example



(j) Scaling example



(k) Flipping example



(l) Noise example

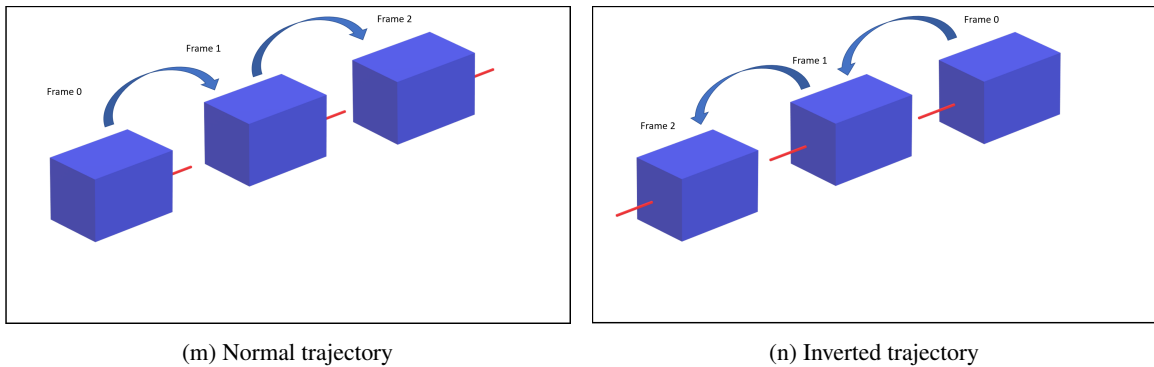


Figure 33: Examples of data augmentation techniques applied to images (a to f) and to bounding boxes (g to n)

In short, we implemented noise, flipping and inverting trajectories (translation and scaling indirectly). Although these techniques are simple, they provide flexibility and cover most of the possible transformations. On the other hand, techniques geared towards image processing were discarded, such as change contrast, saturation, brightness, cropping and colour augmentation. Additionally, rotation was also not implemented.

4.7 PIPELINE IMPLEMENTATION

A tracking algorithm is not usually the end goal, rather just a component for a more complex system. Its real main goal is to keep an object's state up-to-date, however, it must also output said states for every frame.

Keeping that in mind, the tracking data must comprise of the following information:

- *Track ID*: a unique identifier to associate the same object across different frames, given a single 3D point cloud;
- *Class ID*: a unique identifier to associate the object to a predefined class;
- *Centre position*: three coordinates, x, y, z , defining the centre of the object's bounding box;
- *Size*: three coordinates, $\delta x, \delta y, \delta z$, defining the dimensions of the bounding box;
- *Direction*: one coordinate, θ , defining the orientation of the bounding box;
- *Velocity*: three coordinates vector, v_x, v_y, v_z , defining the velocity of the object.

In short, these 12 variables are the base requirement for a simple 3D tracker. With the information mentioned here, it is possible to locate, in space, tracked objects and forecast their trajectories (usually, the two main tasks for a tracker).

4.7.1 Proposed Pipeline

To ensure that the algorithm is able to track this information, this pipeline was designed, as presented in Figure 34. This process can be divided into 5 major components:

- *Object detector*, where 3D bounding box predictions are made, presenting information regarding the object class, its localisation and orientation;
- *Movement prediction*, where future states are predicted, more specifically, objects' position and velocity;
- *Matching tracks*, where tracks (with their updated states) and detections are matched, resulting in 3 sets - matched pairs, unmatched detections, and unmatched tracks;
- *Track start/end*, this module handles unmatched data and is responsible for starting the tracking of new objects and stopping the tracking for objects outside of sensor's range;
- *State update*, last step of the pipeline. Here, it is necessary to update tracking information for both detected and undetected objects. For the latter, it is also necessary to predict its current state, as well as updating confidence, e.g., the state of an object that has not been detected for 3 frames is a lot less reliable compared to an object that has not been detected for 1 frame. For detected objects it is necessary to take into consideration sensor's uncertainty, so it is recommended to update the information as a weighted average between the last stored state and the current/detected state.

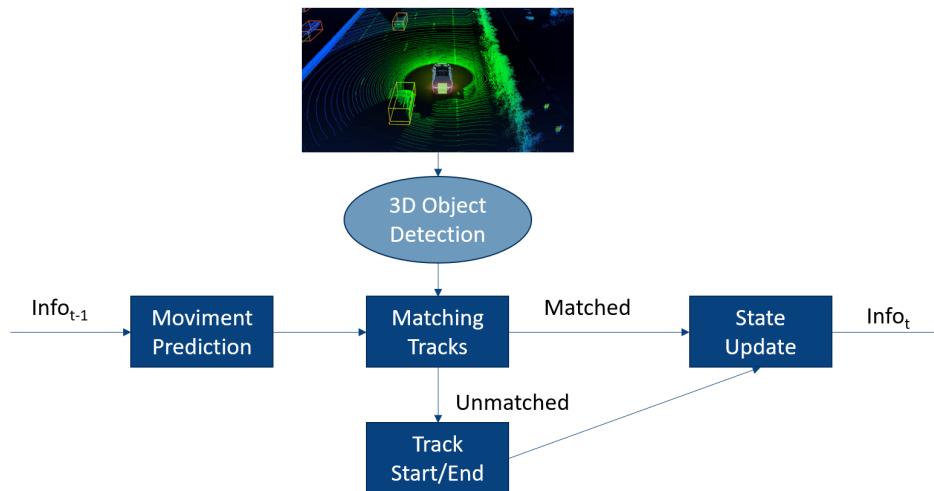


Figure 34: Illustration of the proposed pipeline

4.7.2 First Tests

Thus, to get familiar with this type of input data and the expected output, some *GitHub* repositories [11] were explored for first tests. The main performed tasks were the analysis of the input data format and its representation,

as well as the output format and its respective visual representation. Figure 35 shows that multiple objects have a unique ID and are surrounded by their bounding box.

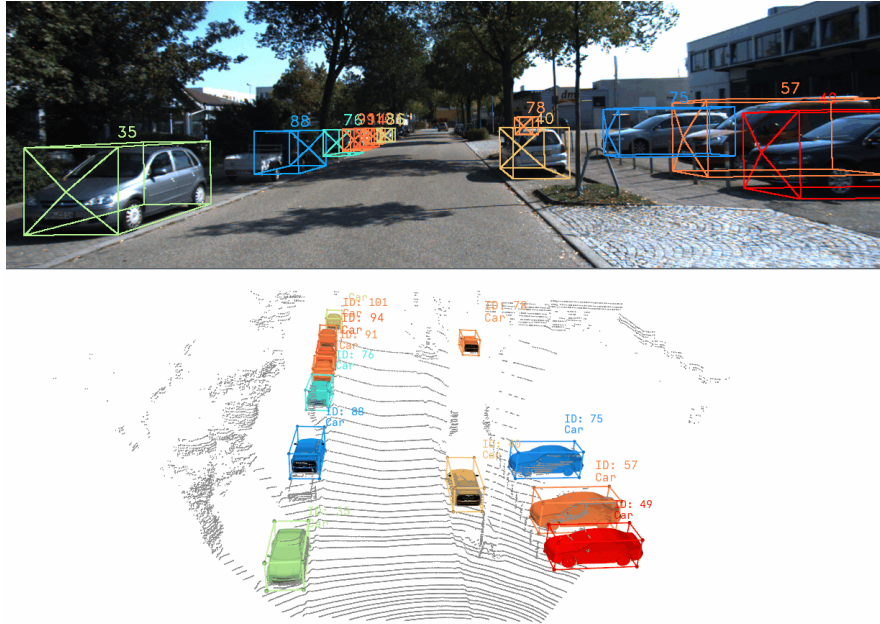


Figure 35: Visual Representation of the expected output for a MOT, based on RGB and LiDAR annotated data (from GIF generated with [11])

As a rule, only LiDAR data will be used for the object detection and tracking tasks, and some of the aforementioned data representation techniques will be explored. In addition to the presented information, the algorithm must also store information for each target, specifically the object bounding box, its unique ID, its velocity, its position and direction (relative to recording test vehicle or absolute values), its size and object class. With all this data, it will be explored the performance/capacity of the algorithm to track objects in a temporal context.

4.7.3 Implementation

In the end, the pipeline's structure differs slightly from what was explained in Section 4.7.1. The pipeline is divided into 4⁴ stages, as depicted in Figure 36: (i) First, the position of all tracked objects is predicted, more specifically its velocity which is added to their latest position; (ii) Then, it is necessary to associate the detections made by the detector with the predicted states to obtain newer information about said tracked objects, as well as maintaining their ID; (iii) Associated objects' state is updated, as well as tracked objects that were not matched. For new objects, i.e., unmatched detections, its state is initialised according only to their detections. Beyond that, it is decided if a previous unmatched detection should be deleted/discarded and if a recently detected and matched detection should be treated as a tracked object; (iv) Lastly, to prevent from having overlapping objects (greater than a certain threshold), a NMS function is used to filter overlapping objects.

4 Excluding the 3D object detector.

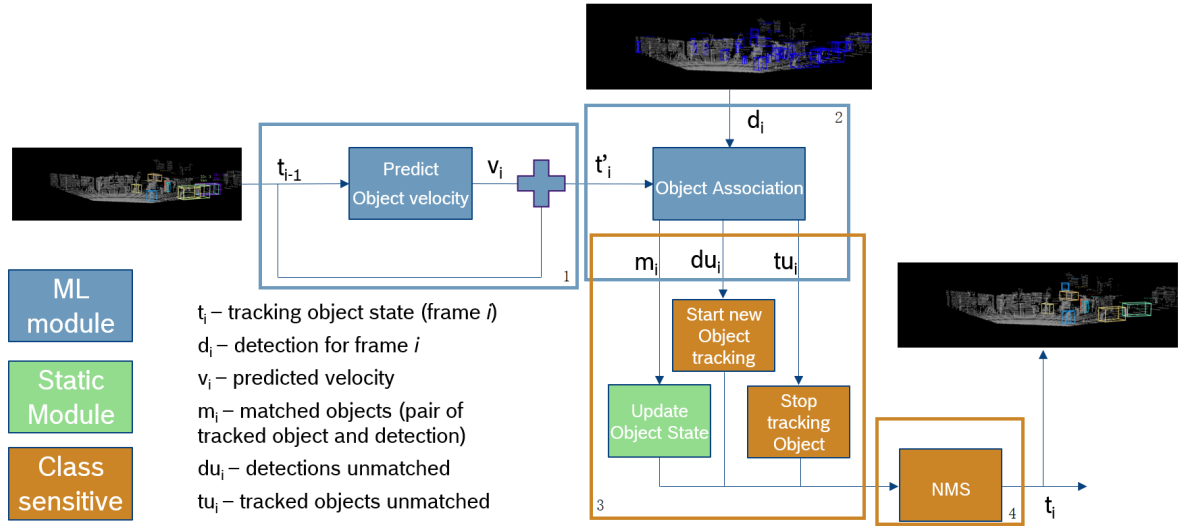


Figure 36: Pipeline representation

The current implementation only works with relative data, i.e., it does not know the absolute position of any objects. This limitation restricts the object tracking performance and prevents the algorithm from working with the surrounding objects' absolute velocity.

4.7.3.1 Prediction

As it has been emphasised, a tracking system must predict the future states of the monitored objects. Its behaviour, or more specifically, the mathematical operations used during prediction, will be discussed first, followed by the model's architecture, ending with its performance and some limitations/problems found during this process.

Predictions are made by predicting the difference between the last and the current state, i.e., using the last N states. After predicting the said difference, it is applied to the last state to obtain the future state, as illustrated in equation 8.

$$s_i = f(i) + s_{i-1} \tag{8}$$

$$f(i) = p(s_{i-N}, \dots, s_{i-1}) = \Delta_i$$

with $f(i)$ representing the object state prediction (corresponds to the predicted variation between frame $i - 1$ and i) in state i (using the N preceding states) and s_i representing the state in frame i .

To perform this prediction, it was implemented a simple DL algorithm, with its structure presented in Figure 37. Although it could used a more complex and consistent algorithm, it was decided that this module should be computationally efficient and simple.

Model: "Prediction Model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 4)	0
cu_dnnlstm (CuDNNLSTM)	(None, 1024)	4231168
dense (Dense)	(None, 1024)	1049600
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 32)	32800
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 4)	132
=====		
Total params: 5,313,700		
Trainable params: 5,313,700		
Non-trainable params: 0		

Figure 37: Prediction model layers and number of parameters

To take advantage over the fact that the objects states are given over a regular period of time, it was decided to use a **LSTM** layer. As mentioned in section 1.2.4.1, these networks are capable of retaining relevant information for a series of data. The number of parameters were obtained through a series of tests to determine what were the best values. The number of units in the **LSTM** layer was calculated using [46]'s formula as a guideline:

$$N_{units} = \frac{N_s}{\alpha * (N_i + N_o)}$$

with N_{units} being the number of units in a **LSTM** unit, N_s , N_i and N_o the number of training samples, input neurons and output neurons, respectively. α corresponds to a scaling factor to prevent overfitting, usually, this value ranges from 2 to 10. In this case, $N_{units} = 1185.75 \approx 1024$ with $\alpha = 5$, the closest power of 2.

Dense layers are applied to support the model learn patterns, based on the features extracted, in order to process and calculate the object prediction values. The last layer is a dense one, which will output 4 values corresponding to the target variables of each object state prediction, i.e., 3 values representing the object's location (x , y , z) and

1 value for its orientation. Dropout layers are used to reduce the probability of overfitting.

Each module was trained separately from the rest of the pipeline, and this division allows to better identify if any module is limiting the overall performance, since we have the individual performance, for the corresponding task, for each module.

As stated in section 4.5, we processed the training data into a specific format. This normalisation creates a consistent input for the DL models. After processing the data, we divided the data from the training split into two parts, one for the learning model's validation (applied to test the model performance with use cases never "seen" during model training process) and another to train the network. The prediction component performance is shown in Table 16.

	IoU	IoU \geq 50%	IoU \geq 70%	IoU \geq 90%	Inference time (ms)
DL approach	85.93%	99.82%	94.30%	37.57%	47ms (batch = 1024)
Linear movement	81.82%	98.78%	86.58%	24.33%	8ms (batch = 1024)

Table 16: Prediction component performance

After analysing Table 16, we can see that the forecasting model achieves an IoU of 85.93% between the forecast and the actual future state. However, upon further analysis, we can see that about 99.81% and 94.30% of the predictions have a 50% and 70% IoU or higher, respectively. However, only 37.57% of the predictions have an IoU greater than 90%, which results in around 5.70% wrong predictions (using the same threshold as from the KITTI detection benchmark), while 37.57% are precise predictions. In comparison, the linear approach is incapable of making high-quality predictions at the same rate as the DL approach.

A reason to justify this low performance is that the predictions made early in the "life" of objects use the initial replicated state, i.e., the first time it detects an object, its state buffer is initialised with N initial states. Another problem is that the current IoU accuracy presents a bottleneck on the overall HOTA.

Even so, we can affirm that the current performance is acceptable, given it is capable of predicting a future state within 70% IoU more than 90% of the time. Furthermore, the remaining modules should be able to mitigate possible problems related to slightly wrong predictions, e.g., when updating objects' states, the resulting information is an average between detection and prediction.

4.7.3.2 Association

In this section, the association component will be explained. The first topic will be its architecture, followed by its results. With that said, the association's goal is to create relationships between tracked objects and new object detections. These relationships will validate if a certain detection corresponds to said tracked object, allowing to update its state, improving the prediction in the next frame.

Model: "Association Model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 7)	0
dense_3 (Dense)	(None, 32)	256
dropout_2 (Dropout)	(None, 32)	0
dense_4 (Dense)	(None, 16)	528
dropout_3 (Dropout)	(None, 16)	0
dense_5 (Dense)	(None, 8)	136
dense_6 (Dense)	(None, 1)	9
Total params: 929		
Trainable params: 929		
Non-trainable params: 0		

Figure 38: Association model layers and number of parameters

Similarly to the prediction module, the association module should present efficient execution times which can be achieved by a simple structure, as described in Figure 38. This module uses slightly different information from prediction's, however, it uses the same data split.

Before feeding any data to the model during training, it is required to predict the tracking state to obtain the information depicted in Figure 32. As such, the first step is to use the information of frame $i - 1$ to predict the future state, producing a predicted state for frame i , imitating what will happen during the pipeline execution. Therefore, association requires a functional prediction module.

This model uses dense and dropout layers for the same reasons mentioned for the prediction component. The output has a dense layer with only 1 value, since this component only deliberates whether it is a correct pair or not. Finally, association performance is described in Table 17.

		Accuracy	Inference time (ms) / batch
Without data augmentation	DL approach	99.97%	44ms (batch = 1024)
	IoU Association (≥ 0.125)	99.75%	286ms (batch = 1024)
Using data augmentation	DL approach	99.94%	44ms (batch = 1024)
	IoU Association (≥ 0.100)	74.46%	286ms (batch = 1024)

Table 17: Association component performance

As can be seen in table 17, the association module achieves a high prediction rate, almost 100%. Initially, it was suspected that the model achieved an almost perfect performance due to the training data being very similar between consecutive frames. A simple static test noting the variations of the object states' between frames could correctly predict nearly all cases, i.e., check if the object's size and position, among others, did not change significantly. However, random noise was applied to the data to delete/dilute these similarities and to make the model more flexible and robust, improving performance for uncertain data. By applying this data augmentation technique, the association DL algorithm continued to achieve similar performance. However, the static test only correctly predicted 74.46% of the cases using the IoU association algorithm, losing approximately 25% compared to the previous value, while the association DL approach was capable of maintaining a similar performance. Contrary to what happened in the prediction module, the association model is faster and more efficient than the deterministic method (i.e., only 44ms compared to 286 ms, for a batch size of 1024). This substantial time difference is due to the heavy mathematical and geometric operations required to calculate the 3D IoU with arbitrary orientation (i.e., not aligned with any referential axis). On the other hand, the difference in accuracy is due to uncertain data (imitating the sensors and detectors errors, among others).

Additionally, when executing the pipeline, association has an extra functionality. To lower the number of possible pairs, in an attempt to lower the time needed for this module, only detections and tracked objects that do not have a significant difference in size and location are accepted as valid pairs. After this filtering, the association module predicts for every detected object if it corresponds to its pair (tracked object), outputting its "confidence". If either a detected or tracked object is pair with more than one pair, then the pair with lower score will be discarded.

4.7.3.3 Track Birth and Track Death

These two modules are responsible for beginning and ending the tracking of every object. The current implementation makes these decisions based on the number of tracked frames, as well as missing frames, i.e., the last time that this object was associated with a detection.

Initially, it was planned a DL approach, however, due to an already time-consuming architecture, it was implemented a static approach, meaning that if an object was associated with N detections then it was considered as a track, and if it disappeared for M frames it was considered as a missing object and, thus, its information was deleted.

Furthermore, a new threshold was defined to enable the tracker to keep objects information in memory without necessarily outputting said object state. This will be useful for objects that are undetected for a long period of time, however, they did not disappears from the scene and can be detected again.

This threshold works exactly like the others, where if the objects is not tracked/associated for K frames but less than M (track death threshold), its information is kept “hidden”, until it surpasses M frames, and then its information is deleted, or appears again and continues as a normal object. These transitions between different states are illustrated in Figure 39.

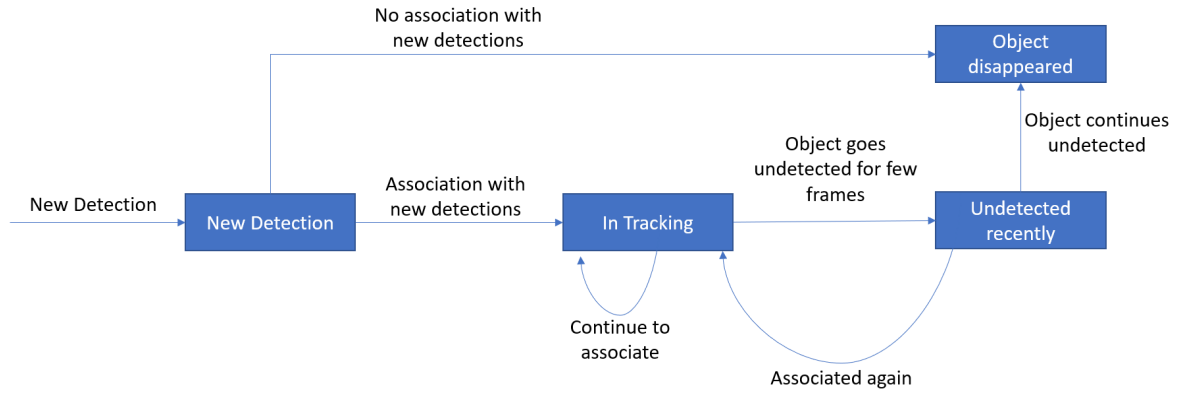


Figure 39: Transition between different states

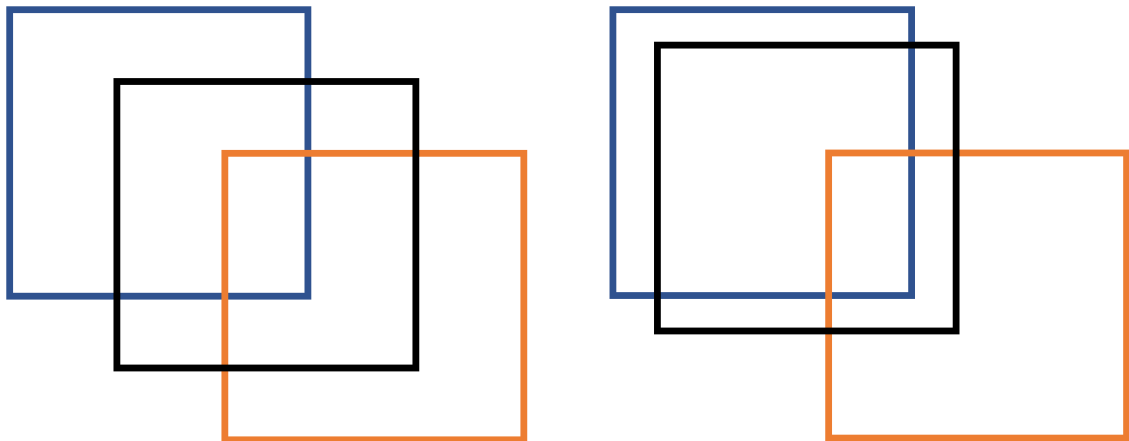
4.7.3.4 State Update

This module is necessary because neither detections nor predictions are perfect, and therefore, it is necessary to lower the risk of aggravating the uncertainty of the objects’ states, as such, it was decided that the best and simpler way was calculating a weighted average between both measures, i.e., object prediction and detection, as depicted in equation 9.

$$s^{i+1} = \alpha d^i + \beta p^i \quad (9)$$

where s^{i+1} , d^i , and p^i represent the future state at frame $i + 1$, the detection and prediction at frame i , respectively. α and β represent the detection and prediction weights, respectively.

As depicted in Figure 40, we can see the differences of using various weights, e.g., in Figure 40a the resulting state will be exactly at the middle of the detected and predicted states. Figure 40b exemplifies the behaviour with a 75% weight for the detection and 25% weight for the tracked state, and we can observe that the updated state is then much closer to its detection.



(a) State update example, using the average of both detection and tracking states (b) State update example, using a weighted average of both detection (75%) and tracking (25%) states

Figure 40: Object state updating with different weights, with blue, orange and black representing its detected, tracked and final state, respectively

4.7.3.5 Non-Maximum Suppression

This module is not included in the original pipeline draft, because the idea behind it only appear after the initial tracking results, where a lot of objects were overlapping leading to lower performance and also a worse representation. Therefore, a function was implemented, similar to what is applied in object detection, that rejects bounding box proposals with high overlap, choosing the one with the highest confidence score. However, this selection is made based on the lifespan of said objects, choosing the oldest (other alternatives were tested, but using the number of tracked frames seem to be the better option).

In short, it iterates over every object in memory and if 2 objects present a high *IoU*, one of them is deleted, leaving the oldest one. This mechanism is illustrated in Figure 41.



Figure 41: Example of NMS's usage

4.8 EVALUATION METRICS

The evaluation of the models' performance is a big factor in AI, as it is usually a rather complex task that depends on several factors and the importance given to each one. Thus, it is necessary to design unbiased techniques to compare results rigorously.

Currently, the state-of-the-art uses a set of metrics to measure and compare an algorithm's performance. The most relevant ones are:

- **Multiple Object Tracking Accuracy (MOTA)** (equation 10), it measures the number of missing and false detections (FN and FP , respectively), and association mistakes (IDS), i.e., ID switches between frames. It compares against all ground-truth labels (GT). This metric has some limitations: (i) detection is more important than association; (ii) is too dependent on the recall value (r); (iii) choosing the best recall value is a time-consuming and complex process; (iv) furthermore, every track is valued equally during evaluation, hence a not so confident track has the same weight as an almost certain track, making evaluation less reliable;

$$MOTA = 1 - \frac{FN_r + FP_r + IDS_r}{GT} \quad (10)$$

- **Average Multiple Object Tracking Accuracy (AMOTA)** (equation 11), an improved version of **MOTA**. The referred improvement is that the metric is the average **MOTA** score for multiple L thresholds, instead of a single threshold like applied in **MOTA**. Despite its improvements, it is also much slower to be computed;

$$AMOTA = \frac{1}{L} \sum_{r \in \{\frac{1}{L}, \dots, 1\}} MOTA_r \quad (11)$$

- **Scaled Average Multiple Object Tracking Accuracy (sAMOTA)** (equation 12), a normalised **AMOTA**, but normalised, ranging between 0 to 1. Unlike **MOTA**, it uses only a portion of the ground-truth (GT) that is limited by the recall value (r). The $\max(0, 1 - \dots)$ ensure that this metric will output a value between the mentioned values, 0 and 1;

$$sMOTA_r = \max\left(0, 1 - \frac{FN_r + FP_r + IDS_r - (1 - r) \times GT}{r \times GT}\right)$$

$$sAMOTA = \frac{1}{L} \sum_{r \in \{\frac{1}{L}, \dots, 1\}} sMOTA_r \quad (12)$$

- **Higher Order Tracking Accuracy (HOTA)** (equation 13), which is currently KITTI's recommended metric, as it provides a better equilibrium between the detection's and association's weights. According to [47],

HOTA measures how well the trajectories of matching detections align and averages this over all matching detections, while also penalising detections that do not match.

$$HOTA = \int_0^1 HOTA_r dr \approx \frac{1}{L} \sum_{r \in \{\frac{1}{L}, \dots, 1\}} HOTA_r \quad (13)$$

$$HOTA_r = \sqrt{DetA_r \cdot AssA_r} = \sqrt{\frac{\sum_{c \in TP} A(c)}{|TP| + |FN| + |FP|}}$$

$$DetA_r = \frac{|TP|}{|TP| + |FN| + |FP|}$$

$$AssA_r = \frac{1}{|TP|} \sum_{c \in TP} A(c)$$

$$A(c) = \frac{|TPA(c)|}{|TPA(c)| + |FNA(c)| + |FPA(c)|}$$

With TPA_r , FNA_r and FPA_r , being the number of **TP**, **FN** and **FP** associations, respectively.

When evaluating for online contexts, it is advised to use online higher order tracking accuracy (Online Higher Order Tracking Accuracy (**OHOTA**)), given that it just uses the information available up to that frame.

Other related metrics are $DetA_\alpha$ and $AssA_\alpha$, these measure the percentage of aligned detections and correct association across frames, averaged for all detections, respectively. Additionally, these can be divided into simpler equations:

$$DetA_r = \frac{DetRe_r \cdot DetPr_r}{DetRe_r + DetPr_r - DetRe_r \cdot DetPr_r}$$

$$DetRe_r = \frac{|TP|}{|TP + FN|}$$

$$DetPr_r = \frac{|TP|}{|TP + FP|}$$

$$AssA_r = \frac{AssRe_r \cdot AssPr_r}{AssRe_r + AssPr_r - AssRe_r \cdot AssPr_r}$$

$$AssRe_r = \frac{|TPA(c)|}{|TPA(c) + FNA(c)|}$$

$$AssPr_r = \frac{|TPA(c)|}{|TPA(c) + FPA(c)|}$$

These metrics, $DetRe$, $DetPr$, $AssRe$, and $AssPr$, are, as their name indicates, the Recall and Precision regarding detection and association, respectively. $DetRe$ and $DetPr$ represent the concepts commonly associated to them, precision and recall, more specifically in object detection. However, $AssRe$ and $AssPr$ are not similar to any well-known concepts, instead $AssRe$ represents the number of tracks necessary for tracking objects, while $AssPr$

represent the number of objects assigned to the same track.

In this case, Recall represents how well trajectories cover ground-truth, while Precision measures how well predicted trajectories associate/follow the ground-truth objects.

In terms of tools to perform this evaluation, many benchmarks use the tool presented in [48].

Furthermore, the analysis of the performance of the detections produced by the tracking algorithm, i.e., the 3D bounding box information produced to track objects, can be evaluated using two concepts, Precision-Recall curve and AP/mAP as explained in section 1.2.5.

4.9 HYPER-PARAMETER TUNING

A hyper-parameter is a parameter whose value influences the learning process. Fine-tuning is the process of accurately adjusting these hyper-parameters to get the best possible performance for a certain task. This technique is often carried out by trial and error, with an objective function guiding the optimisation algorithm in the right direction, with individual hyper-parameters being modified to get different outcomes. In the end, it compares all results to discover which combination of hyper-parameters results in the most accurate model.

The defined hyper-parameters are:

- Frames used for prediction: number of frames processed for predicting future tracked objects' states;
- Association Threshold: minimum value (output of association model) to treat a pair as a match;
- Detection Score Threshold: minimum detection score (output of detection model) to accept a predicted detection;
- Difference Threshold*: the maximum difference between two possible objects, tracked and detected, account for the distance between centre, size difference and angle difference;
- Frames to start tracking*: Number of required matches before an object is "officially" in tracking;
- Frames to pause tracking*: Number of required frames without matches before an object is "officially" not in tracking, however, its information is still in memory;
- Frames to "forget" tracking info*: Number of required frames without matches to delete objects' information. This value can not be lower than the previous parameter;
- IoU Threshold*: Maximum value for a "valid" overlap;
- Prediction weight: applied in the update formula (presented in section 4.7.3.4), e.g., detection weight is always 1, so if prediction's is 1, 0.5 or 0, means $50/50$, $67/33 (= \frac{1}{1+0.5} / \frac{0.5}{1+0.5})$ and $100/0$, respectively.

The hyper-parameters with an asterisk are class-sensitive, i.e., present specific calibrated values according to the objects' class. The reason behind this decision is their detectability, since a car is much more easily detected than a pedestrian and the detection model has less FP in this case, therefore if a car is detected and matched for 2 frames, it is much more probable it is correct. The same can be said when a car is not detected for some frames, which is much more probable the car disappeared.

An optimisation function was implemented to ensure that the best performance was reached, in this case the highest HOTA value. The followed approach was a Grid Search algorithm as explained in [49], where a set of values is defined for each hyper-parameter, and every possible combination is tested. However, one optimisation was made to reduce the search time, given that the Car information does not interfere with other classes', it was decided that every possible combination of non-sensitive-class parameters was tested while the sensitive-class parameters are combined at the same time.

For example, consider 2 non-sensitive-class parameters and 2 sensitive-class parameters for 3 different classes, each has 4 possible values, so for a normal Grid Search there would be $4^2 * (4^2)^3 = 4^8 = 65536$ possible combinations. Through this modification, only $4^2 * \max(4^2, 4^2, 4^2) = 4^4 = 256$ combinations are tested, reducing the execution time by 256 times, and since, normally, the optimisation process ran for about 2 days, without this optimisation it would take 512 days, almost 2 years.

With that said, the best hyper-parameters combination obtained are listed as followed:

- Frames used for prediction: 5;
- Association Threshold: 0.05;
- Detection Score Threshold: 1.0;
- Difference Threshold *: 4.2, 1.1, 2.6 for classes Car, Pedestrian and Cyclist, respectively;
- Frames to start tracking *: 3, 2, 4 for classes Car, Pedestrian and Cyclist, respectively;
- Frames to pause tracking *: 1, 2, 1 for classes Car, Pedestrian and Cyclist, respectively;
- Frames to “forget” tracking info *: 6, 6, 1 for classes Car, Pedestrian and Cyclist, respectively;
- IoU Threshold *: 0.45, 0.62, 0.55 for classes Car, Pedestrian and Cyclist, respectively;
- Prediction weight: 33.3% for prediction, 66.7% for detection.

4.10 OVERVIEW

In short, this chapter served as the base to justify the dataset selection, accompanied by its analysis. It was concluded that the dataset has a diverse representation with multiple classes, even though half of the dataset are objects from the class Car. As noted, there is the expectation that the algorithm will tend to prioritise the

performance of this class, even if it is because of the higher number of training cases, meaning there is a greater probability of improving for the class Car. Furthermore, this analysis did not discover any unusual results, having straightforward conclusions.

Another noteworthy topic is the object detector algorithm to be applied. As explained, the SE-SSD [14] detector had to be replaced during development because it only worked for the Car class, while the chosen one, PV-RCNN [15], performs for 3 (Car, Pedestrian, and Cyclist). It has slightly worse performance but compensates by enabling the tracking of more object classes.

The pre-processing was discussed, including the applied data augmentation process and the theoretical reasoning behind the selection of said techniques.

Afterwards, the proposed solution was presented. It began with the explanation behind the conceptualisation, followed by a brief description of both the overall pipeline and each individual module. Lastly, all hyper-parameters are presented, as well as their optimal values. Metrics are also explained during this chapter, as they are a relevant part of an algorithm's development.

Finally, after presenting the entire developed pipeline, in the next section will be focused on presenting the quantitative and qualitative evaluation analysis behind the proposed algorithm.

EXPERIMENTS AND RESULTS

This section describes all the results obtained in this work and further discussion. The chapter starts with a description of the testing environment, followed by some consideration of the 3D evaluation process. After, the quantitative and qualitative results are presented, ending with the resources requirements and the ablation studies. These experiments were performed in the following environment/configurations:

- **Hardware:**
 - CPU: AMD Ryzen Threadripper 1950X;
 - RAM: 64 GB;
 - GPU: NVIDIA RTX 2080 TI;
 - Storage: SSD Disk with 1TB storage size and 560 MB/s bandwidth.
- **Software:**
 - OS Version: Linux Ubuntu 20.04;
 - CUDA 11.3 and CUDNN 8.2.
 - Python 3.8.10:
 - * NumPy 1.21.0 (for detection, tracking and visualisation);
 - * PyTorch 1.10.0 (for detection);
 - * SpConv 1.0 (for detection);
 - * PCDet 0.5.2 (for detection);
 - * TensorFlow 2.8 (for tracking);
 - * VTK 9.1.0 (for visualisation).

5.1 EVALUATION PROCESS

As explained in section 4.8, the most relevant tracking metrics are [HOTA](#) and [MOTA](#). However, due to [MOTA](#)'s shortcomings and [1]'s preference metric for [HOTA](#), only [HOTA](#) will be evaluated for both 2D and 3D data.

Additionally, as stated before, 3D tracking has a disadvantage because KITTI [1] only benchmarks the values of 2D metrics. With that said, 3D trackers must convert their data to a 2D space. This transformation incurs a gain in performance, however, it does not make up for the complexity of detecting in 3D, thus, benefiting 2D trackers. This conversion is achieved through mathematical operations and it converts the 3D bounding box represented in the 2D space to a 2D bounding box, as depicted in Figure 42. The 2D bounding box will include unwanted pixels and this extra area is expected due to object having width/depth in 3D spaces.

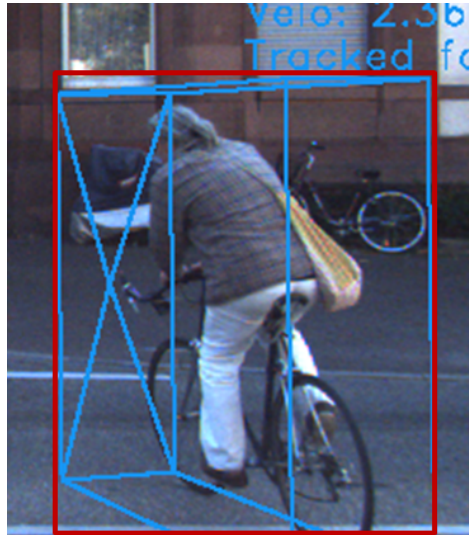


Figure 42: Conversion error of 3D to 2D representation

Thus the conversion of 3D to 2D bounding boxes requires the provided matrices by KITTI [1], where intrinsic and extrinsic matrices are the most relevant. An extrinsic matrix stores the translation and rotation information in the 3D space, while intrinsic matrices store information related to the optical centre and focal length, among others. Essentially, an extrinsic matrix converts data from the world 3D space information to a 3D camera space, while intrinsic converts this information from a 3D camera space to the 2D camera space.

We divided this process into simplified steps. First, it calculates the 8 vertices from the 3D bounding box from the centre point and its dimensions. Next, we multiply these points by the extrinsic matrix resulting in the same 8 points in the camera's frame of reference. Finally, we align these points to the camera orientation using the intrinsic matrix, computing these points as pixel coordinates.

After obtaining the coordinates to all 8 vertices, it is "drawn" a bounding box, where the coordinates are x of both leftmost and rightmost vertices, as well as y of the highest and lowest vertex.

5.2 QUANTITATIVE ANALYSIS

The quantitative analysis will focus on comparing the performance and execution time of the proposed approach and AB3DMOT [8] using the selected tracking metric, **HOTA**. Additionally, it will focus on a study between both

considered detectors, SE-SSD [14] and PV-RCNN [15], regarding their detection performance. Both studies were achieved through the KITTI [1] toolkit. Finally, it ends with some considerations regarding the advantages and drawbacks of a tracking approach versus detection.

In Table 18 contains the results regarding HOTA from both the proposal 3D object tracking algorithm and AB3DMOT [8]. These results show the performance for 3D data, converted 2D, and original 2D data, the latter only for AB3DMOT [8], where * denotes the cases where 2D information is converted from 3D, using the process explained in section 5.1, while bold numbers indicate the best performance, although with comparison is only performed for 3D results.

	HOTA	Car	Pedestrian	Cyclist	Average	Inference time (ms)
Ours	3D	63.72%	44.87%	49.43%	52.68%	107ms
	2D*	72.55%	43.55%	56.90%	57.67%	
AB3DMOT [8]	3D	60.57%	44.09%	44.48%	49.71%	90ms (although [12] states 30 ms)
	2D	70.00%	43.60%	50.49%	54.70%	
	2D*	69.21%	42.90%	50.07%	54.06%	

Table 18: 2D and 3D HOTA results

Looking at these results on Table 18, we can clearly see that our approach achieved vastly improved its 3D results when compared to [8] for class Cyclist, almost 5%, and more than 3% for class Car, and a slightly better performance for Pedestrian, almost 1%.

Additionally, as it was already stated, the 2D performance is higher than 3D's, however, if we look closely at the AB3DMOT's performance for both original and converted 2D results, the performance is lower, which indicates that our tracker's performance, in principle, would slightly increase, leading to a possible higher position in the KITTI Tracking [1] benchmark [12].

Currently, our proposal ranks 5th for LiDAR-only approaches, and 34th in general, both for class Car in KITTI tracking benchmark [12]. For Pedestrian, it ranks 3rd for LiDAR-only approaches, and 23rd in general, as of September 12, 2022.

Table 19 focuses on the bounding boxes quality of both detection and tracking algorithms, i.e., throughout tracking's execution multiple bounding boxes will be predicted and this table tries to compare who the tracking algorithm performs against the input detections.

As it can be seen in Table 19, the bounding boxes generated by the tracking algorithm have lower AP values due to (i) relying in the quality of the detections to which is being compared; (ii) "delaying" the tracking of objects, i.e., only after a required amount of detections, an object can be considered as in "tracking" (threshold to begin tracking in section 4.9); and, (iii) the detector's intermittent detections, i.e., detecting a object in a frame and not detecting in the next, which will delay the tracking even longer.

		Car	Pedestrian	Cyclist
PV-RCNN [15]	Tracking	79.0%	61.8%	66.2%
	Detections	89.2%	72.3%	67.9%
SE-SSD [14]	Tracking	63.2%	-----	-----
	Detections	75.3%	-----	-----

Table 19: 2D and 3D AP results (Moderate difficulty) for both detections' as well as tracking's bounding boxes

In addition, some tests were made where all detections were passed as an object in tracking, and its AP was higher than the original detections. The only difference to the original data was the NMS module, proving that some detections were overlapping where it was not the case.

Although a object tracking approach has some drawbacks, it also has its benefits. While a object detector detects objects at a higher rate, a object tracker has the advantage of being more consistent and reliable, meaning it has a higher resistance to false detections. These false detections only begin to appear when the detector has made an incorrect detection, and, because usually, a tracker does not start to track right away, it will only have consecutive faulty states if the detector wrongfully detects for long periods. In addition, a tracker also has the advantage of calculating objects' velocity, which is essential to have more complete knowledge of its environment.

5.3 QUALITATIVE EVALUATION

This section focus on the discussion about the tracker's qualitative performance. The main topics are the resulting image/representation and the information that comprises each object, cases that the algorithm handles poorly and the possible explanations of that behaviour, and the situations where it works better than expected.

Figure 43 illustrates an example of the visual representation of the output. Both LiDAR and image data are here represented. However, camera data is only used to support visualisation validation. Regarding state information, each object has an assigned object ID and a class. As frames are analysed, the tracker will update their location, orientation, and velocity, among others. Each object has two additional variables representing the age (the number of detected frames) and when it was last detected. These variables only appear where it is relevant, e.g., if an object does not have any information about when they were last detected, it means it was detected and associated in the current frame, hence printing information about the last detection is irrelevant in this case, as it would be 0. However, to better demonstrate each example shown later in this section, this information will always be visible on said examples, as represented in Figure 43. The quantitative analysis highlighted some problems responsible for the tracker's underperformance regarding the classes Pedestrians and Cyclists. However, it does not point out its reason. Therefore, it is required to analyse the produced results to investigate possible problems, e.g., mismatches, wrong predictions, associations and/or detections.

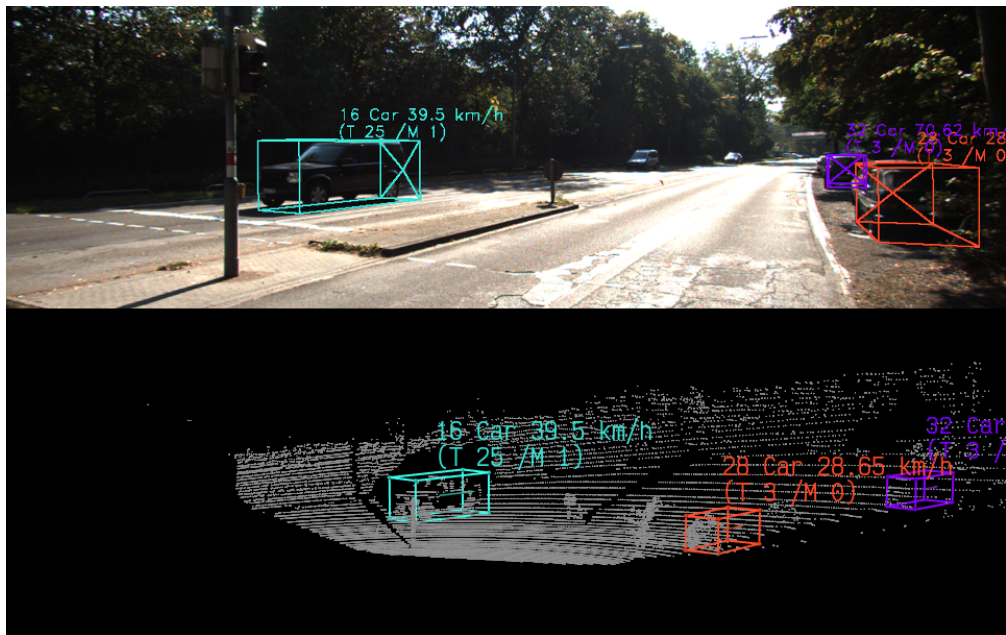


Figure 43: Visualisation example, frame 63 of sequence 14 from KITTI Tracking dataset [1]

After analysing the current result, it is clear that the detector has some patterns which lower the overall performance. Some of these patterns are mistaking parked bicycles and people holding a bicycle as cyclists, as illustrated in Figure 44 and 45. Although these type of mistakes are acceptable, since situations like the ones in depicted in these Figures would leave humans undecided about what classes these highlighted objects belong to.

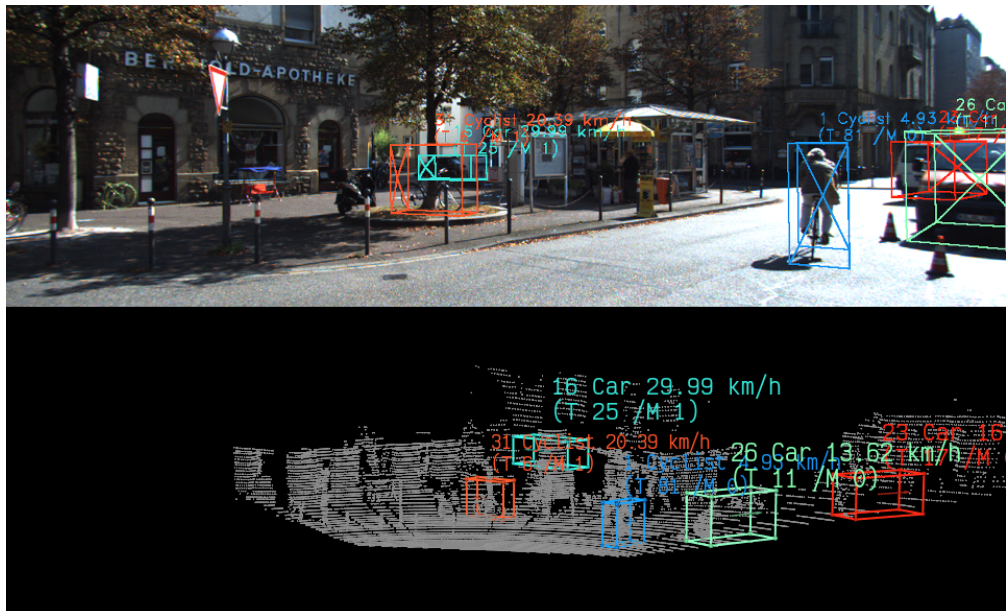


Figure 44: Example of a misclassified detection, labelling a parked bicycle as a Cyclist (ID 31), frame 81 of sequence 0 from KITTI tracking dataset [1]

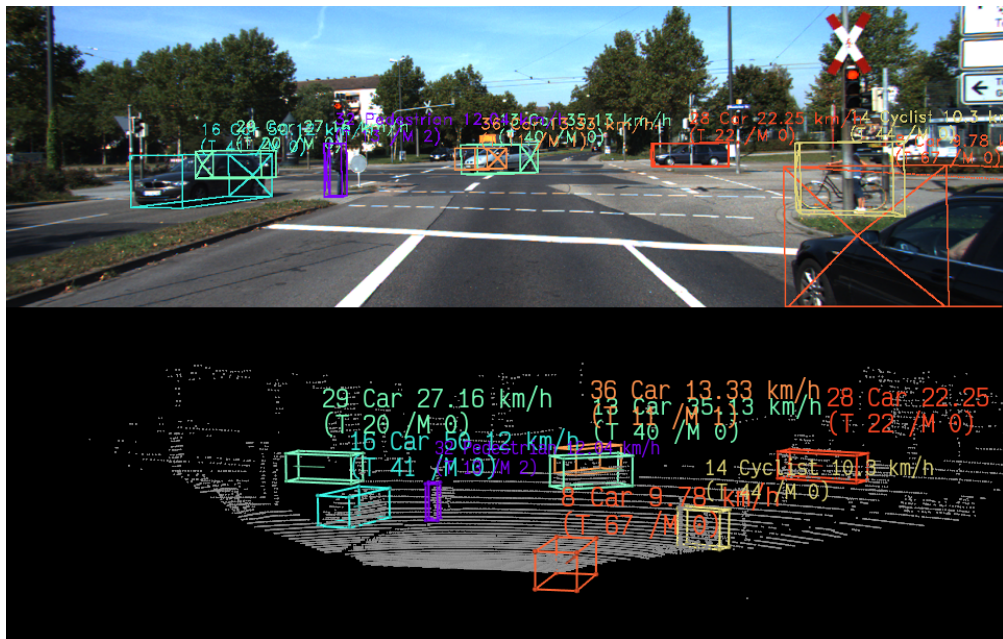


Figure 45: Example of mislabelling a person walking with a bicycle as a Cyclist (ID 14), frame 111 of sequence 2 from KITTI tracking dataset [1]

Another problem in the current situation emerges if the detector begins to make intermittent detections. In other words, an object is rightfully detected but wrongfully disappears in some subsequent frame. In that case, the tracker has some problems signalling the detected object as a real object, because it delays the transition to a state of “tracked-ness”, i.e. surpass the defined tracking threshold explained in section 4.9. Subsequently, another problem also arises, as no detection is being made for that object, its state becomes dependent only on the tracking algorithm which is problematic. When objects are more regularly correctly detected, the results are more consistent and have more precise information about objects’ states.

There is also the problem with detection faraway objects. This problem aggravates for Pedestrians and Cyclist classes due to their dimensions, making them harder to detect using LiDAR, while being still visible in the RGB image. There is also some problems with incorrect detection as shown in Figure 46, where the algorithm “detects” a Car, although it is road poles. Possibly, it is a sign that the detection algorithm is not adapted to all type of environments, as most of the environments do not have street poles dividing the road.

Regarding occluded objects, the tracker showed improvements, as depicted in Figure 47. The car is no longer visible and it has not been detected for X frames, nevertheless, the proposed algorithm is capable to continue its tracking. It is worth noting that a Van is obstructing the LiDAR sensor of obtaining data from the occluded Car. The Van itself is not detected due to the detector presenting difficulties recognising objects from this class. Nevertheless, this test case illustrates the tracker’s ability to continue to track an object even after it is no longer visible in the LiDAR data perspective.

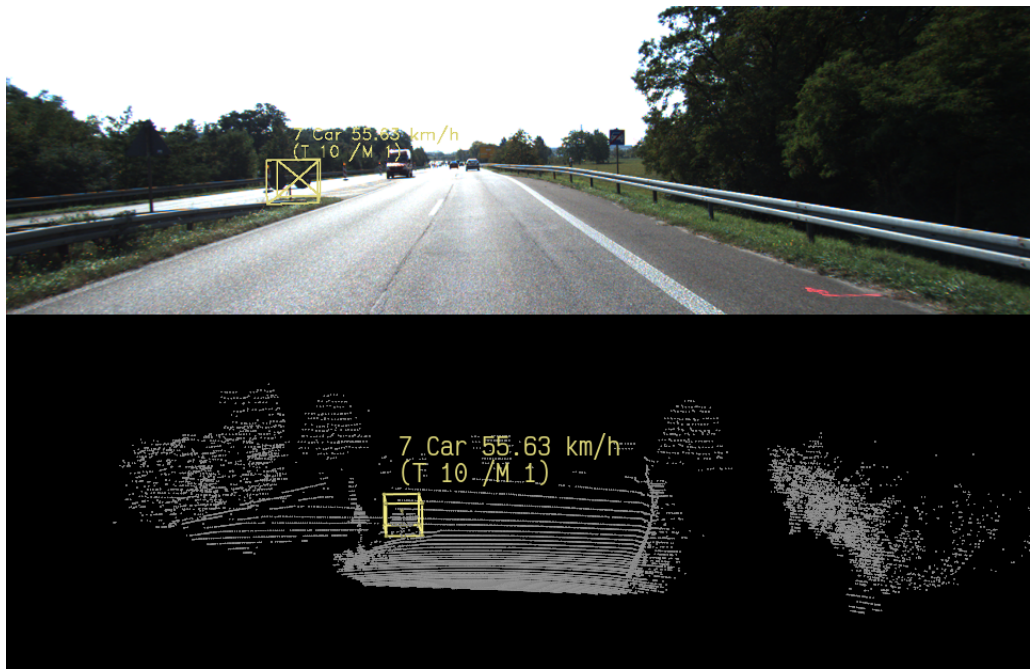


Figure 46: Example of faulty detection where a road sign is predicted as a Car (ID 7), frame 27 of sequence 8 from KITTI tracking dataset [1]

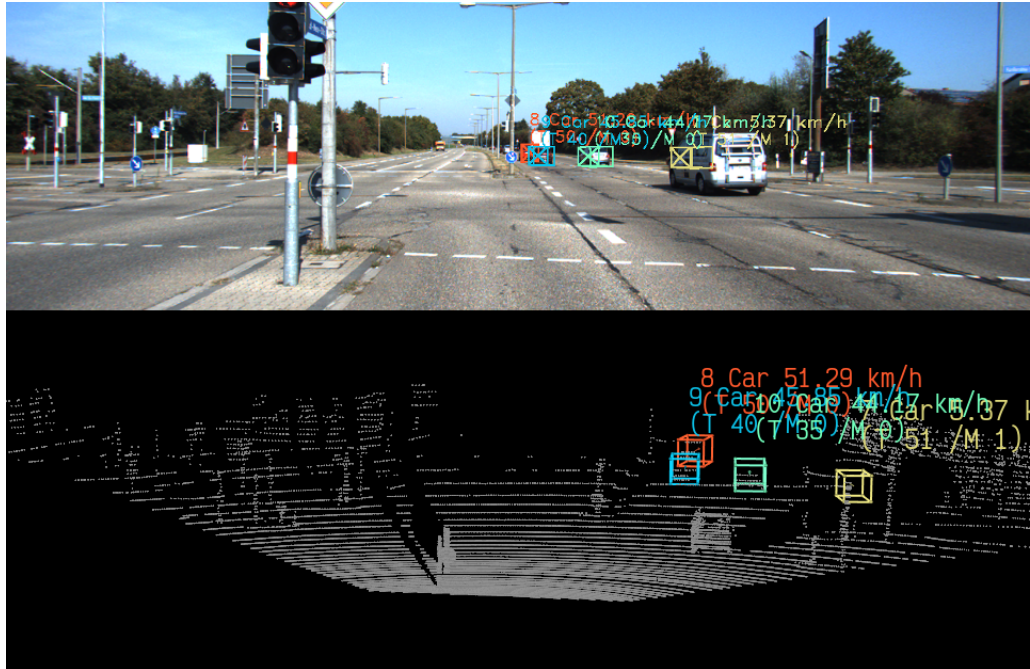


Figure 47: Example of an occluded Car (ID 7) still being tracked, although it is no longer visible in the LiDAR data, frame 53 of sequence 1 from KITTI tracking dataset [1]

5.4 RESOURCE REQUIREMENTS

One of the main points in this type of framework is resources' availability and necessities. For this type of technology to be viable, i.e., run on relatively light-resourced systems, to prevent high costs. As such, a study was conducted to monitor the resources required during its execution. The values are shown in Table 20.

	CPU	GPU (MB)	RAM (MB)	Runtime (FPS)
Ours	4%	40 MB	2094 MB	9.4
AB3DMOT [8]	25%	0 MB	748 MB	11.1

Table 20: Computational resources usage, regarding the tracking task

The proposed architecture achieves a similar performance as stated before in Table 18 while retaining its low resources' usage, although being slightly slower. When comparing both, ours has an insignificantly CPU and GPU usage, specifically, 4% of CPU usage and 40MB of GPU memory which is negligible. The main downside is the RAM's usage, which almost tripled. However, 2094 MB is still considered acceptable and doable in today's standards.

Even though, these usages are promising, given we did not use any type of compression method to decrease resources' usage and increase the number of FPS. It is worth noting that these compression processes usually maintain a similar performance, while improving the computation efficiency to execute these algorithms.

5.5 ABLATION STUDY

The ablation study will be conducted in this section to determine each component's impact on the overall performance. These experiments will be conducted by replacing or removing elements from the proposed solution with ones that operate differently yet have the same purpose.

There were made tests to compared the performance of SE-SSD [14] performance for both KITTI [1] detection and tracking datasets. Its performance dropped from 89% to 82%, this difference indicates their algorithm was optimised for the detection dataset, proving that its detector's performance does not reflect its capability to perform well on uncontrolled and new scenarios.

No. Test	Type			Classes				Inference time (ms)
	Association	Prediction	Features	Car	Pedestrian	Cyclist	Average	
1	DL	DL	NMS + Ass. Filter	63.72%	44.87%	49.43%	52.67%	107 ms
2	DL	DL	NMS	63.06%	33.81%	46.65%	47.84%	105 ms
3	DL	DL	Ass. Filter	62.50%	44.62%	49.43%	52.18%	99 ms
4	DL	Linear	NMS + Ass. Filter	63.01%	42.34%	49.45%	51.60%	58 ms
5	DL	Static	NMS + Ass. Filter	50.95%	35.17%	36.89%	41.00%	63 ms
6	Distance	DL	NMS + Ass. Filter	36.81%	24.51%	20.26%	27.19%	53 ms
7	IoU	DL	NMS + Ass. Filter	61.33%	45.42%	49.39%	52.05%	73 ms
8	IoU	DL	NMS	61.26%	45.31%	49.70%	52.09%	86 ms
9	IoU	Linear	NMS + Ass. Filter	60.24%	44.57%	49.12%	51.31%	18 ms

Table 21: Ablation experiments and respective HOTA performance

The current DL approach, test 1, achieves the highest performance out of every conducted test, as stated in Table 21. The closest performing test uses the DL prediction and association modules and the association filter as described in section 4.7.3.2, discarding only the NMS module.

Linear prediction, tests 4 and 9, surprised as it was capable of achieving slightly lower performances than DL's prediction. Another finding was that no or static movement, i.e., without predicting the objects' paths, and distance-based association, in tests 5 and 6, achieve considerably worse performances.

Another surprise is the filtering during association to ignore distant or distinct objects since it resulted in a improvement of almost 5% comparing to test 1 and 2. However, when comparing tests 7 and 8, this filter does not improve the overall performance, since IoU already takes into account the objects' distance and size decreasing this filter's relevance. IoU-based association proved to be a reliable alternative, achieving better results for Pedestrians and similar for Cyclist, yet its performance for Car is significantly lower. Which leads to believe that the DL association benefits from having input data with mainly training samples from class Car.

When comparing NMS's and association filter's tests (2 and 3) with the test 1, their lower performances indicate that these features (NMS and Association Filter) contributed to the overall improvement.

Regarding inference time per frame, the DL-less approach is the fastest, however its performance is almost 1.5% lower, specially for class Car, where it is almost 3.5% lower. By comparing the difference of execution times, it is also perceptible that both association and prediction modules run at similar times, around 45 to 50 milliseconds each.

CONCLUSIONS, LIMITATIONS AND FUTURE WORK

AD has become a industry in itself making AV the main focus of most automotive companies. This increasing interest is being fulfilled by the advances in AI and DL. To accelerate this pursuit, multiple steps are being considered, better perception/awareness of the environment, by either using improved and accurate sensors which will improve object detection, improving the hardware and system's infrastructure used, or even improving the perception algorithms applied in the vehicles, such as object tracking. This allows the AV to act with more confidence, more precisely and faster. In this sense, AV need to be developed using data from the integrated sensors to infer and reason to perform the perception task.

MOT is one of the critical components of these perception modules, responsible for following multiple target objects in its environment. Prediction about future states are useful since the ego-vehicle is capable of understanding and anticipating probable events to act accordingly.

Thus, the main goal of this research work was to develop a LiDAR-based 3D MOT. The use of only 3D point cloud data can be justified by the spatial information that it is provided by the LiDAR sensor, enable the access to more precise information. Additionally, RGB data could unlock further performance improvements due to the combination of dense information inherent to images with the already refereed spatial precision of LiDAR.

The side goals of this work regarded the study of multiple state-of-the-art approaches and benchmark its performance and its computational cost. After an extensive analysis, the proposed and implemented approach is: (i) dependant on the quality of the detections provided by the detector; (ii) presents minor problems regarding the detection of cyclists, where it mixes up bicycles in various conditions with cyclists; and, (iii) is able to track objects even if occluded or no longer in the sensors' search area.

In short, the performance of the proposed solution relies on the object detector performance. If the detector fails/malfunctions for several frames, the algorithm eventually will lose precious data, however the same can be said for every approach.

Finally, experiments shown that the NMS component and introduces an improvement in comparison to a NMS-less approach. Also, DL prediction achieves better results than linear prediction, the equivalent to association was confirmed, where DL association performed better than IoU and distance approaches.

6.1 LIMITATIONS

As mentioned throughout this document, several limitations were discovered, ranging from the pipeline's performance to data synchronisation. Although not every issue resulted in incompleting implementations, 3 main issues raised the most concern. The first one is related to the execution time, one of the most relevant characteristics of any autonomous driving application. The last two are related to the quality of the achieved results. Beyond expecting results within a short time frame, it is also necessary to achieve quality results. These two factors govern the choice of the best-suited systems for any area, especially in a field with such high risk as autonomous driving.

As shown in Table 15, each detection takes 80ms and since the LiDAR sensors gather 10 frames per second, each frame takes 100ms, leaving only a window of 20ms to perform each tracking iteration. In addition, each model's inference (both prediction and association) take about 40ms, hampering to perform this task in real-time, with using any compression mechanisms and/or optimisations for the neuronal networks.

The detector itself is not perfect as it has many limitations. The most noticeable difficulty is the inability to distinguish bicycles for cyclists, yet it is acceptable due to being similar in terms of shape and shows the model recognises a pattern in cyclists (cyclists ride bicycles). Additionally, there are some frames where it detects objects wrongfully for multiple frames, as exposed in section 5.3. Another problem is the inability to detect Van or Truck objects, even though the detector accepts them as a Car object. The detector is capable of detecting these type of objects, however this happens very rarely. Nevertheless, the tracker also showed improvements in terms of being aware of objects that are no longer visible and, therefore, detectable. This ability is one of the main advantages of the temporal approaches, such as tracking, which enables the system to use/access further information.

6.2 FUTURE WORK

After concluding this thesis, there are some interesting future implementations directions focused mainly on improving the current performance and/or execution time.

The easiest way to achieve the first goal is to either (i) apply a better detector, improving the results by raw quality of bounding boxes or more consistent detections. One of the easier paths to achieve such performance improvement is by using more data to train the detector. As mentioned above in section 6.1, although the detector accepts Van objects as Car, it presents problems detecting said objects and most of the times that type of vehicles are simply ignored. Another advantage of having a more extensive annotated dataset is the possibility of expanding the definition of each class, p.e., the class Cyclist, as mentioned, only regards people riding a bicycle, however, if a person is driving a motorcycle it is simply ignored. This type of situation should not exist as Motorcyclist are a vulnerable and frequent "road-user"; (ii) improve the prediction model, valuing more its prediction, making the approach more resistant to subpar detections. Despite that, there is not much room for improvement without converting detections' relative to absolute coordinates. This conversion would be valuable in situations where the

recording test vehicle is turning and/or accelerating, i.e., the vehicle's movement is not linear nor constant. Absolute coordinates can be calculated from odometry data, in this case, GPS data. The vehicle's position at frame 0 is considered as the origin, i.e., the point $(0, 0, 0)$. From then on, every detected object's position is transformed to the origin coordinate system. While this transformation is easy for objects detected at frame 0, since its origin is already $(0, 0, 0)$, for objects from future frames, the vehicle has moved and, therefore, its origin changed. Thus, it is necessary to calculate the changes from frame i to 0, in order to convert relative coordinates to an absolute frame of reference. Additionally, this approach would also allow to calculate objects' absolute velocity, by calculate the displacement between two frames, which is relevant in the AV's context. Additionally, velocity could also be used as input in the prediction task. Currently, this approach has some problems that were not solved due to the short time frame of this work.

To achieve the latter objective, faster execution times, the simplest way would be to use TensorRT which has mechanisms available to optimise the algorithm's inference, such as pruning and knowledge distillation, among other.

This process is achieved by either reducing the number of layers, restricting their size, making modification to the variables class (to use more efficient alternatives) or some other mechanism.

It allows to perform tasks up to 4 times faster [50]. Nevertheless, there is a trade-off, i.e., balancing between model prediction accuracy and performance. Due to their potential, these techniques are used in multiple areas, such as AI and autonomous machines (for example, AV), among others.

Another possible technique is to use additional information during the detection phase, for example, using 2D information to complement the LiDAR data (this could be also applied to the association phase) and extract poses to improve association, among others.

Analysing the performance for multiple ranges is an interesting study to perform, as it would allow us to understand if the tracking deteriorates itself with distance. Faraway objects have the same value as close objects still correctly tracking close objects is far more relevant because of the safety and collision risks. To conduct this evaluation, we need to adapt the current evaluation pipeline to filter objects for certain ranges yet aware of objects close to the limits. For example, if we define a 0-50 metres range, an object 51 metres away that is detected 50 metres away should be included in the current range even though it is inside the defined area.

Another experiment is to test the model using detections at 5 FPS instead of 10FPS, i.e, skipping a detection every two detections, to make the algorithm both online and real-time, and testing if there is any loss of performance.

Finally, it would be interesting to test the current solution in embedded hardware, to debug any problem that might arise, due to the uncontrolled environment. This step would be relevant to make this work more complete, as it would serve as a proof of concept.

BIBLIOGRAPHY

- [1] A. Geiger, P. Lenz, and R. Urtasun. **Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite**. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
- [2] Society of Automotive Engineers. **SAE Levels of Driving Automation™ Refined for Clarity and International Audience**. [Link.](#), 2021.
- [3] F. Galbusera, G. Casaroli, and T. Bassani. **Artificial intelligence and machine learning in spine research**. *JOR SPINE*, 2:e1044, 02 2019. doi: 10.1002/jsp2.1044.
- [4] T.Shin. **An Intuitive Explanation of Gradient Descent**. [Link.](#), Jan 2020.
- [5] R. Padilla, S. L. Netto, and E. A. B. da Silva. **A Survey on Performance Metrics for Object-Detection Algorithms**. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 237–242, 2020. doi: 10.1109/IWSSIP48289.2020.9145130.
- [6] S. Sharma, J. A. Ansari, J. K. Murthy, et al. **Beyond Pixels: Leveraging Geometry and Shape Cues for Online Multi-Object Tracking**. *CoRR*, abs/1802.09298, 2018. URL <http://arxiv.org/abs/1802.09298>.
- [7] E. Baser, V. Balasubramanian, P. Bhattacharyya, et al. **FANTrack: 3D Multi-Object Tracking with Feature Association Network**. *CoRR*, abs/1905.02843, 2019. URL <http://arxiv.org/abs/1905.02843>.
- [8] X. Weng and K. Kitani. **A Baseline for 3D Multi-Object Tracking**. *CoRR*, abs/1907.03961, 2019. URL <http://arxiv.org/abs/1907.03961>.
- [9] H.-K. Chiu, J. Li, R. Amrus, et al. **Probabilistic 3D Multi-Modal, Multi-Object Tracking for Autonomous Driving**. *CoRR*, abs/2012.13755, 12 2020. URL <https://arxiv.org/abs/2012.13755>.
- [10] H. Wu, Q. Li, C. Wen, et al. **Tracklet Proposal Network for Multi-Object Tracking on Point Clouds**. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1165–1171. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/161. URL <https://doi.org/10.24963/ijcai.2021/161>. Main Track.
- [11] hailanyi (Github user). **3D-Multi-Object-Tracker** and **3D-Detection-Tracking-Viewer**. [Link.](#) and [Link.](#), 2021.
- [12] **Object Tracking Evaluation (2D bounding-boxes)**. [Link.](#)

- [13] S. Chen, B. Liu, C. Feng, et al. **3D Point Cloud Processing and Learning for Autonomous Driving**. *CoRR*, abs/2003.00601, 2020.
- [14] W. Zheng, W. Tang, L. Jiang, et al. **SE-SSD: Self-Ensembling Single-Stage Object Detector From Point Cloud**. *CoRR*, abs/2104.09804, 2021. URL <https://arxiv.org/abs/2104.09804>.
- [15] S. Shi, C. Guo, L. Jiang, et al. PV-RCNN: point-voxel feature set abstraction for 3d object detection. *CoRR*, abs/1912.13192, 2019. URL <http://arxiv.org/abs/1912.13192>.
- [16] H. Caesar, V. Bankiti, A. H. Lang, et al. **nuScenes: A multimodal dataset for autonomous driving**. In *CVPR*, 2020.
- [17] **3D Object Detection Evaluation 2017**. [Link](#).
- [18] WHO. **Global Status Report on Road Safety 2018**. [Link.](#), 2018.
- [19] NHTSA. **Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey**. [Link.](#), 2015.
- [20] Honda. **Honda Develops World's First 'Collision Mitigation Brake System' (CMS) for Predicting Rear-end Collisions and Controlling Brake Operations**. [Link.](#), 2003.
- [21] Defense Advanced Research Projects Agency. **The Grand Challenge**. [Link](#).
- [22] H. Lim Si Min and A. Taeihagh. **Algorithmic Decision-Making in AVs: Understanding Ethical and Technical Concerns for Smart Cities**. *Sustainability*, 11(20), 2019. ISSN 2071-1050. doi: 10.3390/su11205791. URL <https://www.mdpi.com/2071-1050/11/20/5791>.
- [23] S. J. Russell and P. Norvig. **Artificial Intelligence: A Modern Approach (2nd edition)**. Prentice Hall, December 2002. ISBN 0137903952.
- [24] National Research Council. **Funding a Revolution: Government Support for Computing Research**. The National Academies Press, Washington, DC, 1999. ISBN 978-0-309-06278-7. doi: 10.17226/6323. URL <https://www.nap.edu/catalog/6323/funding-a-revolution-government-support-for-computing-research>.
- [25] S.Schuchmann. **History of the Second AI Winter**. [Link.](#), May 2019.
- [26] P. McCorduck. **Machines Who Think (2nd edition)**. A. K. Peters, 2004.
- [27] Tom M. Mitchell. **Machine Learning**. McGraw-Hill, New York, 1997. ISBN 978-0-07-042807-2.
- [28] IBM Cloud Education. **What is deep learning?** [Link](#).
- [29] Pérez-Enciso and Z. Laura. **A Guide for Using Deep Learning for Complex Trait Genomic Prediction**. *Genes*, 10:553, 07 2019. doi: 10.3390/genes10070553.

- [30] S. Dobilas. **LSTM Recurrent Neural Networks – How to Teach a Network to Remember the Past.** [Link.](#), Feb 2022.
- [31] Prabhu. **Understanding of Convolutional Neural Network (CNN) – Deep Learning.** [Link.](#), Mar 2018.
- [32] C. C. Chatterjee. **Implementation of RNN, LSTM, and GRU.** [Link.](#), Jul 2019.
- [33] IBM Cloud Education. **What are Neural Networks?** [Link.](#), Aug 2020.
- [34] T. Shin. **A Beginner-Friendly Explanation of How Neural Networks Work.** [Link.](#), Jun 2020.
- [35] V. Breen, M. Silva, B. Valente, et al. **Using multiple regression, Bayesian networks and artificial neural networks for prediction of total egg production in European quails based on earlier expressed phenotypes.** *Poultry Science*, 11 2014.
- [36] **Artificial Neural Networks.** [Link.](#), Feb 2017.
- [37] **Gradient Descent.** [Link.](#)
- [38] R. Qian, X. Lai, and X. Li. **3D Object Detection for Autonomous Driving: A Survey.** *CoRR*, abs/2106.10823, 2021.
- [39] H. Wu, W. Han, C. Wen, et al. **3D Multi-Object Tracking in Point Clouds Based on Prediction Confidence-Guided Data Association.** *IEEE Transactions on Intelligent Transportation Systems*, pages 1–10, 2021. doi: 10.1109/TITS.2021.3055616.
- [40] H. Kuhn. **The Hungarian Method for the Assignment Problem.** *Naval Research Logistic Quarterly*, 2, 05 2012.
- [41] G. McLachlan. **Mahalanobis Distance.** *Resonance*, 4:20–26, 06 1999. doi: 10.1007/BF02834632.
- [42] S. Shi, X. Wang, and H. Li. **PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud.** *CoRR*, abs/1812.04244, 2018.
- [43] Y. Liao, J. Xie, and A. Geiger. **KITTI-360: A Novel Dataset and Benchmarks for Urban Scene Understanding in 2D and 3D.** *arXiv.org*, 2109.13410, 2021.
- [44] **Waymo Open Dataset: An autonomous driving dataset**, 2019.
- [45] **Dog Image.** [Link.](#)
- [46] K.Eckhardt. **Choosing the right Hyperparameters for a simple LSTM using Keras.** [Link.](#), Nov 2018.
- [47] J. Luiten, A. Ošep, P. Dendorfer, et al. **HOTA: A Higher Order Metric for Evaluating Multi-Object Tracking.** *International Journal of Computer Vision (2020)*, 2020. doi: 10.1007/s11263-020-01375-2.
- [48] J. Luiten and A. Hoffhues. **TrackEval.** [Link.](#), 2020.

- [49] J. Rohan. **Grid search for model tuning.** [Link.](#), Dec 2018.
- [50] A. Chaturvedi. **Understanding Nvidia TensorRT for deep learning model optimization.** [Link.](#), Oct 2020.

