



HAL
open science

Driving scene understanding from automotive-grade sensors

Florent Bartoccioni

► **To cite this version:**

Florent Bartoccioni. Driving scene understanding from automotive-grade sensors. Artificial Intelligence [cs.AI]. Université Grenoble Alpes [2020-..], 2023. English. NNT : 2023GRALM018 . tel-04193785

HAL Id: tel-04193785

<https://theses.hal.science/tel-04193785>

Submitted on 1 Sep 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : MSTII - Mathématiques, Sciences et technologies de l'information, Informatique

Spécialité : Informatique

Unité de recherche : Laboratoire Jean Kuntzmann

Interprétation de scène de conduite à l'aide de capteurs automobile

Driving scene understanding from automotive-grade sensors

Présentée par :

Florent BARTOCCIONI

Direction de thèse :

KartEEK ALAHARI

Chargé de recherche HDR, INRIA CENTRE GRENOBLE-RHONE-ALPES

Directeur de thèse

Patrick PEREZ

Ingénieur HDR, Valeo.ai

Co-directeur de thèse

Rapporteurs :

Vincent LEPETIT

PROFESSEUR ASSOCIE, ENPC ParisTech

Alexandre ALAHI

PROFESSEUR ASSISTANT, EPFL

Thèse soutenue publiquement le **28 avril 2023**, devant le jury composé de :

Vincent LEPETIT

PROFESSEUR ASSOCIE, ENPC ParisTech

Rapporteur

Alexandre ALAHI

PROFESSEUR ASSISTANT, EPFL

Rapporteur

Mathieu CORD

PROFESSEUR DES UNIVERSITES, Sorbonne University

Examinateur

AuréliE BUGEAU

PROFESSEUR DES UNIVERSITES, Université de Bordeaux

Examinatrice

Jean-Sébastien FRANCO

MAITRE DE CONFERENCES, Grenoble INP

Examinateur

Cordelia SCHMID

DIRECTEUR DE RECHERCHE, INRIA Centre de Paris

Présidente

Invités :

KartEEK ALAHARI

CHARGE DE RECHERCHE HDR, INRIA CENTRE GRENOBLE-RHONE-ALPES

Patrick PÉREZ

INGENIEUR HDR, Valeo.ai

Éloi ZABLOCKI

INGENIEUR DOCTEUR, Valeo.ai



Abstract

Autonomous driving technology has the potential to revolutionize transportation, making it safer, more efficient, and more accessible for everyone. However, achieving full autonomy requires a complex system that can perceive and understand the environment in real-time. In the context of mass-produced passenger cars, automotive-grade sensors, such as cameras and few-beam LiDARs, are crucial components of such a system. Despite their ability to provide rich and diverse information about the scene, these sensors also present significant challenges. For instance, few-beam LiDARs may suffer from noise and sparsity, while estimating the scene geometry from cameras only is difficult. To overcome these challenges, this thesis proposes two novel approaches to leverage automotive-grade sensors for driving scene understanding.

The first part of the thesis revisits the task of depth estimation from a monocular camera; a key feature of autonomous systems that are often equipped with multiple independent cameras. Existing methods either rely on costly LiDARs (32 or 64 beams), or only on a monocular camera signal, which present various ambiguities. To circumvent these limitations, we propose a new approach that combines a monocular camera with a lightweight LiDAR, such as a 4-beam scanner, typical of today’s mass-produced automotive laser scanners. Our self-supervised approach overcomes scaling ambiguity and infinite depth problems associated with camera-only methods. It produces a rich 3D representation of the environment without requiring ground truth during learning. Moreover, as our approach leverages sensors typical of automated cars on the public market, it finds direct applications in Advanced Driver Assistance Systems (ADAS).

The second part of this thesis presents a transformer-based architecture for vehicle and driveable area segmentation in Bird’s-Eye-View (BEV) from multiple cameras. A setup particularly challenging as both the geometry and semantic of the scene must be extracted from 2D visual signals alone. Although BEV maps have become a common intermediate representation in autonomous driving, real-time prediction of these maps requires complex operations, such as multi-camera data extraction and projection into a common top-view grid. These operations are usually performed with error-prone geometric methods (e.g., homography or back-projection from monocular depth estimation) or expensive direct dense mapping between image pixels and pixels in BEV (e.g., with MLP or attention). The proposed model addresses these issues by using a compact collection of latent vectors to deeply fuse information from multiple sensors. This results in an internal representation of the scene that is reprojected into the BEV space to segment vehicles and driveable areas. We also provide evidence that the model also enables accumulating knowledge about the scene over time directly in the latent space, paving the way for efficient reasoning and planning.

The proposed models are validated on real-world datasets and prototype cars, demonstrating the potential of utilizing automotive-grade sensors for driving scene understanding. By addressing the challenges associated with these sensors, our approaches provide a viable path towards their deployment in autonomous driving systems.

Résumé

La technologie de conduite autonome a le potentiel de révolutionner les transports, les rendant plus sûrs, plus efficaces et plus accessibles à tous. Cependant, atteindre une autonomie totale nécessite un système complexe capable de percevoir et de comprendre l'environnement en temps réel. Dans le contexte des voitures grand public produites en série, les capteurs de qualité automobile, tels que les caméras et les LiDAR à peu de faisceaux, sont des composants essentiels d'un tel système. Malgré leur capacité à fournir des informations riches et diverses sur la scène, ces capteurs présentent également des défis importants. Par exemple, les LiDAR à peu de faisceaux produisent un signal spatialement parcimonieux et bruité, tandis que l'estimation de la géométrie de la scène uniquement à partir de caméras est difficile. Pour surmonter ces défis, cette thèse propose deux approches innovantes pour tirer parti des capteurs de qualité automobile pour la compréhension des scènes de conduite.

La première partie de la thèse revisite la tâche d'estimation de la profondeur à partir d'une caméra monoscopique, une caractéristique clé des systèmes autonomes souvent équipés de plusieurs caméras indépendantes. Les méthodes existantes reposent soit sur des LiDAR coûteux (32 ou 64 faisceaux), soit uniquement sur un signal de caméra monoscopique, présentant diverses ambiguïtés. Pour contourner ces limitations, nous proposons une nouvelle approche qui combine une caméra monoscopique avec un LiDAR léger, tel qu'un scanner à 4 faisceaux, typique des scanners laser automobiles produits en série aujourd'hui. Notre approche auto-supervisée surmonte l'ambiguïté de mise à l'échelle et les problèmes de profondeur infinie associés aux méthodes basées uniquement sur les caméras. Elle produit une représentation 3D riche de l'environnement sans nécessiter de vérité terrain pendant l'apprentissage. De plus, notre approche, tirant parti des capteurs typiques des voitures automatisées sur le marché public, trouve des applications directes dans les Systèmes d'Aide à la Conduite Avancés (ADAS).

La deuxième partie de cette thèse présente une architecture basée sur un transformer pour la segmentation des véhicules et des zones praticables en vue aérienne (BEV) à partir de plusieurs caméras. Une configuration particulièrement difficile car à la fois la géométrie et la sémantique de la scène doivent être extraites des signaux visuels 2D uniquement. Bien que les cartes BEV soient devenues une représentation intermédiaire courante dans la conduite autonome, la prédiction en temps réel de ces cartes nécessite des opérations complexes, telles que l'extraction de données multicaméra et la projection dans une grille en vue de dessus commune. Ces opérations sont généralement effectuées avec des méthodes géométriques sujettes aux erreurs (par exemple, l'homographie ou la rétroprojection à partir de l'estimation de la profondeur monoscopique) ou un mappage dense direct coûteux entre les pixels de l'image et les pixels en BEV (par exemple, avec MLP ou système d'attention). Le modèle proposé traite ces problèmes en utilisant une petite collection de vecteurs latents pour fusionner profondément les informations de plusieurs capteurs. Cela résulte en une représentation interne de la scène qui est reprojétée dans l'espace BEV pour segmenter les véhicules et les zones praticables. Nous fournissons également des preuves que le modèle permet d'accumuler des connaissances sur la scène au fil du temps directement dans l'espace latent, ouvrant la voie à un traitement et une planification efficaces.

Les modèles proposés sont validés sur des ensembles de données du monde réel et des

voitures prototypes, démontrant le potentiel d'utilisation des capteurs de qualité automobile pour la compréhension des scènes de conduite. En relevant les défis associés à ces capteurs, nos approches offrent une voie viable vers leur déploiement dans les systèmes de conduite autonome.

Acknowledgements

Those close to me know my love for food and a PhD journey is a recipe that calls for a mix of ingredients: mentorship, collaboration, friendship, inspiration, and so much more. Every person I have crossed paths with has added a special flavor to this mix. Now let's indulge in this feast of gratitude, akin to a chef, but without the heat and occasional kitchen disaster. And if anyone's name was inadvertently missed, please chalk it up to my sleep-deprived brain.

First, I want to acknowledge my supervisors, Patrick Pérez from Valeo.ai, and Karteek Alahari from Inria. Patrick, I would like to express my immense gratitude for the trust you've placed in me. Our chance encounter at Technicolor in Rennes has been nothing short of a pivotal moment in my research journey. I still feel privileged to have witnessed firsthand the transformation from an ambitious idea that is valeo.ai to a thriving research hub of more than twenty bright minds. Your unwavering dedication and passion in nurturing this environment have been an inspiring spectacle, offering a profound model of leadership and vision.

Karteek, my heartfelt thanks for your support, thoughtfulness and patience throughout my PhD. You have been there for me every step of the way (all the way to New Zealand!), guiding me through the complex maze of research and providing support whenever needed. Thank you, deeply, for your advice and guidance on this path.

I also can't be grateful enough to you, Matthieu Cord, for teaching me to stand on my own two feet - both literally and figuratively. I credit my strong legs and knowledge of their preservation to you ;). Your weekly nudges have been like the zesty touch needed to elevate the flavor of this recipe. Your constant encouragement has made a significant difference in my thesis. Above all, beyond your high standards and drive for excellence, I always felt "bienveillance", a quality that goes beyond kindness and goodwill, and I want to express my sincere thanks for embodying it.

Another individual who has left an indelible mark on my research journey is Eloi Zablocki. Eloi, your exceptional communication and writing abilities has been nothing short of invaluable. Through your mentorship, I've learned how to articulate my ideas more clearly, structure my thoughts effectively, and enhance my overall communication skills. You are an exceptional individual who has had an immeasurable impact on my journey. I loved working with you, and I am glad I can continue doing it. Thank you for everything, Eloi.

To all four of you, I thank you for putting together such an esteemed jury for my defense. I cannot say how proud and honored I am to defend in front of researchers whom I admire and whose works have influenced academic path. To the jury members, thank you for the time you've dedicated to review my manuscript and attend the defense. The opportunity to present my work before such a distinguished panel is something I hold in the highest regard.

My heartfelt gratitude to everyone at Inria, for making this more than just an academic expedition. You all created an atmosphere of camaraderie and fun that's second to none. Alexandre, TheGreatZouzou, often my first hello in the morning, you will be remembered for our friendly discussions but above all for turning a simple hike into a Spartan race :'). Mathilde, Valentin and Minttu, I've been fortunate to share an office with such wonderful colleagues. Mathilde, thank you for the vibrant and positive energy, also thanks to you my "to-do list before I die" now counts diving in the shimmering waters of Mexico. Minttu, thank you for sharing the best hiking apps I could find, I couldn't become a full-fledged "grelou" tho. Valentin, I wish you good luck in San-Francisco, your constant cheerfulness

has always brought a lively atmosphere to our shared office. Theo, our discussions were a treat, even if you did forget your swimsuit at a pool event, which still puzzles me! Romain, thank you for introducing me to time bomb, I will remember to never trust you again ;) (even though we won the last round). Juliette, our 'SNCF ticket dealer' and PhDs' 'Gentile Organisatrice', I still have to share the focaccia recipe btw! Zhiqui, Michael, Emmanuel, Loic, and Thomas R., you were our own little pocket of sunshine, bringing joy and kindness to every interaction. Houssam, for always making our minds work around ethical issues during coffee breaks and initiating a chess frenzy among the team. Julien Zhou, for your unlimited arsenal of GIFs and your part in forming the Losers team. Hadrien, I admire your sportsmanship at LoL, a feat to be admired. Enrico, you brought the Italian touch to our team, supplementing our chats with some "gesticolare" and perfect imitations of the typical French "euh, je sais pas". Thomas Lucas, thank you for unveiling the world of split keyboards to me. Thomas de Mine, your corn-starch 'trick' for caccio e pepe was cheeky, but I've got to admit, despite oneself, it worked wonders. Nathalie and Julien, for the amazing retreats you organized. Masha, who else could make me discover bubble tea and their.. intriguing... relationship with goats? Ricardo self-proclaimed "la machina", the master of chaos, thank you for always making things more fun ;). Bruno, with your singing, I'm expecting your name on a billboard any day now! Good luck with your future entrepreneurial ventures. To Timothee, Jules, Leva, Lina, Bulent, Khue, Margot, Heesung, Pierre, Jocelyn, Nikita, Nassim, and Pierre-Louis, I'm grateful to have shared this journey with you all.

During these, not three but four, years at Valeo.ai, I had the privilege to meet extraordinary interns, PhD students, engineers, and researchers who added a unique flavor to my PhD. Thank you, Andrei, our resident wise owl, for your endless knowledge that is like an all-you-can-read buffet of papers. Thank you for your kindness, time, and expertise. Thank you for accompanying and advising me on the technology transfer projects. I loved collaborating with you since my internship :). Bjorn, Corentin, Laura, Léon, Victor L., Antonin, and Loïck, sharing coffees, beers, and 'planches' in Paris with you guys was like a mini holiday in the middle of the month for me! Mickael, thank you for our lengthy debates about world models and generative models, I "forecast" that there are many more to come ;). Oriane, thank you for being our local social event planner and for teaching the proper elbow-lifting etiquette in Prague, I treasure those six-month memories. Alexandre, Gilles, and Renaud, our 3D expert team, I am grateful for your insights, wisdom, willingness to help, and the time taken to answer all of my questions. Spyros, thank you for our discussions, your dedication to the GPUs, keeping them always fired up, was both admirable and a bit scary! Tuan-Hung, the grandmaster of domain adaptation and the last samurai of the "protein team", Eduardo, thank you for making our team a colorful and lively mosaic with your vibrant personality. Cédric, I can't wait to put our discussions on coding practices into action. Yihong, thank you for sharing your knowledge on detection, tracking and trajectory forecasting. Thank you, Pascal and Ouardia, your help with administrative tasks during my internship and at the start of my PhD was like a safety net, always ready to catch me. Alain and Ouafa, you've taken the torch, and it's comforting to have you on the team. Thibaut, Sophia, and Dennis, it's a pleasure to have met you, I wish you all the luck with your internships and future endeavors. Thank you, Serkan, for making me discover the field of norms and regulations. Gabriel, the Python sensei, thank you for guiding me through the labyrinth of Python best practices and setting up our dear cluster. Thank you, Tristan, for always keeping our cluster operational. Thank you to Heidi, Simon, David, Victor B., Arthur, Charles, Maxime, Antoine, Himalaya, Huy, and Maximilian, I'm glad our paths intertwined in this journey.

I would also like to thank Ahmet, thank you for your kindness and assistance during my internship in Prague. Your support made a huge difference and I appreciate it deeply. Thank you Ondra, Giorgos, and the "flu" people for welcoming me during these months. Also thank you, David Pichardie, for your trust and tutoring at ENS Rennes, for opening up the world of scientific research to me. Thank you to Killian, with whom I completed my 'prépa', pursued my journey at ENS Rennes, and collaborated on my first research project. I extend my gratitude to all the professors I had the pleasure of learning from.

I am also incredibly grateful for the circle of close friends and family that have stood by me on this journey. Nico, Tristan, my childhood comrades, no matter the miles between us, our bond remains unaltered. Your presence is my safe haven, every problem dissipates in the freshness of your company. Alexandre, my comrade from 'prépa', your unwavering support over the years has been a significant influence in my life. Our motto, 'Food and Dive', sums up our shared joys perfectly. Caro, thank you for being my accidental match-maker who brought the best person into my life. Thank you also for being the person who gets us moving, Jeanne and I, homebodies that we are ;)! Thank you to the Grenoble's *doggy team* ;), Alizée, Matthias, Alicia and Yann, you are an endless local source of happiness.

To my family, the ones who instilled in me a drive to excel, who enveloped me with love, care, laughter, who molded me into the person I am today, and inspired me with their lives. Your influence and my thankfulness are beyond words.

Jeanne, my partner in all things, we've shared every single part of this journey. Your presence at my side has been a constant reassurance during the tough times, a safe harbor amidst the storm. My very own French Riviera. The highs and the lows, they have all been made better because you've been by my side. And in times of success, sharing our joyous highs, you've always been my cherished co-celebrant. You already know everything that I would like to say but, thank you for being an integral part of this work, thank you for bringing Scarlett in my life, thank you for your love and for your support <3.

And finally, a heartfelt thank you to my parents, the unfaltering beacon of support and love that has illuminated every step of my life, for whom words will never be enough to express my gratitude. This thesis is for you.

Contents

Abstract	i
Résumé	iii
Acknowledgements	v
1 Introduction	1
1.1 Motivation and context	1
1.2 From ADAS to autonomous driving	1
1.3 Typical sensors of autonomous vehicles	3
1.4 AD software stack	5
1.5 Contributions	7
1.5.1 Outline	7
1.5.2 Publications and implementations	8
2 Deep learning applied to autonomous driving	9
2.1 Tasks and public datasets	9
2.1.1 Perception	9
2.1.2 Motion prediction	11
2.2 Challenges	14
2.2.1 Modalities of very different natures	14
2.2.2 Robustness to faulty sensor and visual impediments	17
2.2.3 Training with few or no annotations	18
2.2.4 Transformation of representation	19
2.3 Representing a dynamic scene	20
2.3.1 In the camera image plane	20
2.3.2 In the 3D space	22
2.3.3 In the Bird’s-Eye-View	22
2.3.4 Implicit	23
2.4 Positioning	25
3 Monocular metric depth estimation with a few-beam LiDAR	27
3.1 Introduction	28
3.2 Related work	31
3.3 LiDARTouch framework	34
3.3.1 Depth network	34
3.3.2 Self-supervision objectives	35
3.3.3 Pose estimation	37
3.4 Experimental protocol	37
3.4.1 Dataset and evaluation metrics	37
3.4.2 Notations, ablations and external baselines	38
3.5 Influence of a touch of LiDAR	39
3.5.1 Ablation of LiDAR	39

3.5.2	LiDAR self-supervision variants	41
3.6	Comparison against related works	42
3.7	Alleviating the infinite-depth problem	44
3.7.1	Catastrophic Distance Rate (CDR) metric	44
3.7.2	Quantitative analysis	45
3.7.3	Qualitative analysis	46
3.8	Implementation details	47
3.9	Conclusion	49
3.9.1	Summary of contributions	49
3.9.2	Perspectives	49
4	Latents and Rays for an Implicit Scene Representation	51
4.1	Introduction	51
4.2	Related work	52
4.2.1	BEV semantic segmentation	52
4.2.2	Incorporating geometric priors in Transformers	54
4.3	LaRa: Our Latents and Rays Model	54
4.3.1	Input modelling with geometric priors	55
4.3.2	Building latent representations and deep fusion	55
4.3.3	Generating BEV output from latents	56
4.4	Experiments	56
4.4.1	Evaluation details	56
4.4.2	Comparison with previous works	58
4.4.3	Extension to the driveable area segmentation task	59
4.4.4	Model ablation and sensitivity to hyper-parameters	59
4.4.5	Study of attention	61
4.4.6	Qualitative Results	64
4.5	Extension to temporal modelling	65
4.5.1	Additional modules	65
4.5.2	Results	65
4.6	Conclusion	66
4.6.1	Summary of contributions	67
4.6.2	Perspectives	67
5	Conclusions and future directions	69
5.1	Conclusions and discussions	69
5.2	Future directions	69
5.2.1	Handling multiple types of cameras and different intrinsics.	69
5.2.2	Leveraging Simulation	70
5.2.3	Learning an implicit representation of the world	70
A	Monocular metric depth estimation with a few-beam LiDAR	73
A.1	Overfitting to input LiDAR	73
A.2	Ablation of LiDAR: further analysis	73
A.3	Dilated LiDAR	74
A.4	Pose scaling is critical when using a PnP pose estimation with photometric loss only	78
A.5	Poor performances for ACMNet, NLSPN and S2D when trained with P+IMU	78

B Latents and Rays for an Implicit Scene Representation	79
B.1 Output embedding	79
B.2 Additional attention qualitative analysis	79

List of Figures

1.1	The VaMP driverless car	2
1.2	Sensors' specificities.	4
1.3	Sensor setup of the Valeo Drive4U prototype.	5
1.4	Schematic illustration of the classic modular pipeline of autonomous driving	6
2.1	Examples of annotations from the WoodScape dataset	10
2.2	Common representations for the motion prediction task.	13
2.3	Illustration of a simple Convolutional Neural Network (CNN)-based classifier	14
2.4	Illustration of fully convolutional networks	15
2.5	Representations of point clouds	16
2.6	Schematic illustration of the self-attention layer	16
2.7	Depth estimation artefact from a glare	17
2.8	Illustration of the Perceiver IO architecture	20
2.9	Illustration of a self-supervised learning system for depth estimation	21
2.10	Illustration of the resolution problem that the Bird's-Eye-View representation faces	24
3.1	Different LiDAR densities	30
3.2	Illustration of the self-supervised image-only depth estimation framework	31
3.3	Illustration of the fully-supervised depth completion framework	32
3.4	Overview of our LiDARTouch learning framework	33
3.5	Depth networks with different image-LiDAR fusion strategies	34
3.6	Comparison of different supervision schemes for the ACMNet architecture	43
3.7	Selecting vehicles to compute the CDR metric	45
3.8	Plot of the CDR metric for various thresholds τ	46
3.9	Mitigation of the infinite-depth problem	47
3.10	Qualitative comparison of LiDARTouch with other existing frameworks	47
4.1	Overview of our LaRa architecture	54
4.2	Qualitative results on complex scenes	60
4.3	Sensitivity study of LaRa to hyper-parameters	61
4.4	Input-to-latent attention study	62
4.5	Input-to-latent attention study — influence of the input embedding	63
4.6	Measuring the attention consistency across cameras	63
4.7	Qualitative results on complex scenes	64
4.8	Overview of Lara temporal	66
4.9	LaRa meta-architecture and extension to other tasks	67
A.1	Predictions of a model overfitting to LiDAR input	74
A.2	Statistics for the depth and pose outputs over a training run with a pose network and ('P+L ₄ ') supervision	75
A.3	Loss values over a training run with a pose network and ('P+L ₄ ') supervision	76

A.4	Visual difference between vanilla and dilated LiDAR	77
B.1	Six input camera images coming from the 360-degree camera rig of nuScenes	80
B.2	Input-to-latent attention study — average over latents	80
B.3	Input-to-latent attention study — average over heads	81
B.4	Input-to-latent attention study — all the attention heads of a latent vector .	82
B.5	Input-to-latent attention study — all the latent vectors for an attention head	83

List of Tables

2.1	Overview of autonomous driving datasets	12
3.1	High-level positioning of LiDARTouch vs depth estimation and depth completion methods	28
3.2	Pose estimation ablation	40
3.3	Variants comparison of the LiDAR self-supervision	41
3.4	Comparison against monocular depth estimation methods	42
3.5	Comparison against supervised and naively self-supervised depth completion schemes	43
4.1	Intersection-over-Union (IoU) for vehicle segmentation on nuScenes	59
4.2	Driveable area segmentation	59
4.3	Ablation study for the input and output query embedding	60
4.4	Impact of ray embedding on performance	61
4.5	Impact of ray embedding on cross-camera attention consistency	64
4.6	Results of temporal integration	66

List of Abbreviations

AD autonomous driving 4, 5, 9, 13, 14, 18, 19

BEV Bird's-Eye-View xiii, 7, 13, 15, 18–20, 22–24, 51, 65–67, 71

CNN Convolutional Neural Network xiii, 7, 14, 15, 18, 70

FPN Feature Pyramid Network 15

MDP Markov Decision Process 13

RL Reinforcement Learning 13

Chapter 1

Introduction

1.1 Motivation and context

With the advent of powerful computing technologies and advances in sensor systems, the development of autonomous vehicles gained momentum in the 2010s. Major car manufacturers, technology companies, and start-ups began investing heavily in the development of autonomous vehicles, and the technology quickly moved from the realm of science fiction to a tangible reality.

One of the key drivers of this growth was the recognition that autonomous vehicles had the potential to dramatically improve driving safety and efficiency. By removing human error from the equation, autonomous vehicles could help to reduce the number of accidents and fatalities on the roads; make transportation more efficient, by reducing traffic congestion and allowing vehicles to travel closer together at higher speeds; help improve the mobility of people with disabilities and elderly people not able to drive anymore.

On safety alone, the facts are striking: in 2016, about 1.4 million people died in car accidents and 50 million were injured. Roads are now the eighth leading cause of death for all age groups, surpassing HIV/AIDS, and the leading cause of death for children and young adults aged between 5 and 29 years [WHO, 2016]. From a societal perspective, it has been estimated that in the USA alone, motor vehicle crashes in 2019 cost \$340 billion in economic activity, and nearly \$1.4 trillion of societal harm when considering the loss of life and decreased quality of life from injuries [NHTSA, 2019]. In addition, more than 90% of these road crashes are caused by human error. According to the US National Highway Traffic Administration, in 2017, alcohol-impaired-driving fatalities accounted for 29% of overall deaths, distraction for 8.5%, and speeding-related fatalities for 26% of total fatalities [NHTSA, 2017]. Similar statistics hold in a non-negligible number of other countries [OECD, 2017]. Although this does not come without issues and limits, having all cars with automatized (fully or partially) driving could significantly reduce the number of road accidents and their severity.

1.2 From ADAS to autonomous driving

Advanced Driver Assistance Systems (ADAS) are ubiquitous in today's vehicles and have already helped save lives and prevent injuries since 1950 [Galvani, 2019]. Early ADAS focused on vehicle stabilization systems, e.g., anti-lock braking (ABS) and traction control (TCS). Nowadays, with the advances of embedded electronics, it is common to find vehicles providing functionalities such as adaptive cruise control, obstacle detection or lane-following.¹

¹<https://caradas.com/adas-statistics> - [Last accessed on 2023-1-2]



FIGURE 1.1: Interior of the VaMP driverless car [Mercedes-Benz, 2016], one of the first actual autonomous car that completing in 1995 a 1,758 kilometres (1,092 miles) trip in (almost) complete autonomy

Building on the early developments in ADAS, the PROMETHEUS project (PROgramMe for a European Traffic of Highest Efficiency and Unprecedented Safety, 1987–1995) [Dickmanns, 2002], a research programme headed by car manufacturers from six European countries, pioneered autonomous driving. This project involved over forty research establishments as well as automotive and industrial partners. Its aims were diverse, each raising important research questions: improving road circulation without building new roads; increasing safety and reducing the number of accidents despite an increasing number of vehicles; enabling maximum mobility while boosting efficiency; achieving all these goals while simultaneously preserving the environment. To this end, the research project was formulated in seven sub-projects, covering a wide range of problems such as “driver assistance by computer systems”, “methods and systems of artificial intelligence”, “custom hardware for intelligent processing in vehicles” and “traffic scenario for new assessment and introduction of new systems”. This project paved the way for further research and development in the field, and today many of the technologies that were developed as part of the project are now being used in our vehicles including lane keeping, collision avoidance, autonomous cruise control.

The PROMETHEUS project concluded with a 1,758 kilometres (1,092 miles) trip in 1995 from Munich, Germany to Odense, Denmark, in one of the first truly autonomous cars (fig. 1.1). Driving in free lanes, convoy driving with distance keeping depending on speed, and lane changes left and right have been performed autonomously from two black-and-white video-cameras (one facing forward and the other backward) and a small neural network only. These operations were demonstrated in heavy traffic and on highways; sometimes at speeds above 175 kilometres per hour (109 mph) on the Autobahn, a highway without speed limits. Overall, the trip was realized with almost no human intervention, achieving a 95% autonomous driving.²

Starting from 2004, the US Defense Advanced Research Projects Agency (DARPA) sponsored a series of autonomous vehicle competitions known as the DARPA Grand Challenge [DARPA, 2004]. The goal of these challenges was to encourage the development of autonomous vehicles and raise public awareness of the technology. In 2004, 15 teams from universities and private companies participated in the competition: no vehicle could complete the 142-mile course through the Mojave Desert. Despite the many technical challenges, the challenge continued to evolve over the next few years, becoming more complex

²<https://www.youtube.com/watch?v=I39sxwYK1EE> - [Last accessed on 2023-02-08]

and more closely resembling real-world driving scenarios (driving with passing cars on the opposite lane, avoiding static obstacles, braking at a stop line). As autonomous vehicles move closer and closer to becoming a reality on our roads and highways, the impact of the DARPA Grand Challenge cannot be overstated: it helped to bring autonomous driving out of the lab and raise public awareness.

Since then, investment and development in this area have skyrocketed, with major automotive companies and technology startups entering the market with their own unique approaches. More recently, in 2016, the National Highway Traffic Safety Administration (NHTSA) adopted the six-level classification of automated driving systems introduced by the Society of Automotive Engineers (SAE):

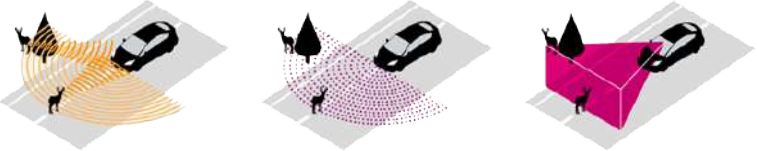
- **Level 0** – The human driver does all the driving.
- **Level 1** – Automated system(s) on the vehicle can take control over one functionality (e.g., adaptive cruise control, ABS, automatic emergency brake assist, and lane-keeping or centering). The human driver constantly oversees operations.
- **Level 2** – Automated system(s) on the vehicle can take control over multiple functionalities to aid the driver (e.g., highway assist, autonomous obstacle avoidance, and autonomous parking). The human driver constantly oversees operations.
- **Level 3** – Automated system(s) can perform all aspects of the driving task under specific circumstances, but the human driver must be ready to take back control when the automated system requests (e.g., highway chauffeur).
- **Level 4** – Automated system(s) can perform the driving task in certain environments and under certain conditions without any human supervision (e.g., automated valet parking).
- **Level 5** – The automated system can perform all driving tasks, under all conditions (the steering wheel and pedals commands become optional)

Each level can itself be further decomposed with what is called an ‘operational design domain’ (ODD). ODDs refer to specific driving scenarios and conditions under which autonomous vehicles (AVs) are designed and thoroughly tested to operate safely and effectively. These domains may include specific road types, weather conditions, speed limits, and other parameters that define the operating limits of the AV. Driving on a highway in broad daylight is very different from driving on a mountain road at night, both in terms of terrain complexity and visual aspects. How to adapt to different domains and being robust to harsh weather conditions are two essential research directions for AVs to operate in all conditions.

1.3 Typical sensors of autonomous vehicles

From the smallest level of automation, automated vehicles must evolve in complex and changing environments, posing the necessity to acquire information about the scene state (e.g., street layout, interactions and types of agents, etc.). To this end, automated vehicles are equipped with a wide range of sensors.

Two of the most critical sensors utilized in autonomous vehicles for object detection and avoidance are radar and LiDAR. Radar operates by emitting radio waves that bounce off objects in the environment and measuring the reflected signals. The velocity of one object is estimated thanks to the Doppler effect, while the distance orientation of objects



	(1) Radar	(2) 64-beam LiDAR	(3) Camera
Range	250m	150m	250m
Resolution	Average	Goods	Excellent
Works in dark	Excellent	Excellent	Mediocre
Works in very bright light	Excellent	Excellent	Good
Works in snow/fog/rain	Excellent	Average	Poor
Provides color and texture	Poor	Poor	Excellent

FIGURE 1.2: Overview of the sensors typically powering the system of autonomous vehicles and their specificities. Source: Delphi

can be derived from the correlation between emitted and received signals (if the radar has enough antennas). This information, still very noisy at this stage, can be further processed to detect other vehicles, pedestrians, and other objects on the road. LiDAR, on the other hand, uses laser light to create a 3D map of the environment. The LiDAR sensor emits pulses of laser light and measures the time it takes for the light to return after hitting an object. Often, LiDAR are equipped with an internal rotating mirror which enables scanning in a full surround manner (one revolution is called a ‘sweep’). This information can be used to construct a highly accurate 3D representation of the environment, allowing the system to reason about the surroundings of the vehicle.

Along with radar and LiDAR, cameras are also a critical, and nowadays ubiquitous, component in the perception system of autonomous vehicles. One of the primary advantages of using cameras over other sensors is their ability to capture colour and texture; necessary to identify objects such as road signs and traffic lights, specifically designed for human visual perception. Cameras generally also provide a high level of resolution, which is essential for the detection of small or distant objects.

Overall, when it comes to building an autonomous vehicle, selecting the right set of sensors is an important decision. Each type of sensor has pros and cons that must be carefully considered (see [fig. 1.2](#) for an overview). Radar, for example, is excellent at detecting objects at a distance and can work well in adverse weather conditions, but has limited resolution and is subject to interferences, making it difficult to accurately identify objects. LiDAR, on the other hand, provides high-resolution information about the environment, but it can be expensive and can struggle in rainy conditions (the laser scatters on droplets of waters). Cameras are an attractive option due to their ability to detect colour and texture, but they can be impacted by weather conditions and tend to struggle in low-light conditions. In this regard, how to get the best from each sensor in order to optimize the efficiency as well as the safety and overall performance of the autonomous system is a research topic of great importance for the autonomous driving (AD) community.

Ultimately, the choice of sensors depends on the specific requirements of the application the autonomous vehicle is being used for. Typical vehicles currently on the robot-taxi market, such as those developed by [Cruise](#) and [Waymo](#), employ cutting-edge sensor technologies and feature highly redundant systems to ensure safety and reliability. These vehicles are equipped with a vast array of sensors, including multiple cameras, radars, and dense LiDARs (32 or 64-beam). This over-engineered approach results in vehicles that are

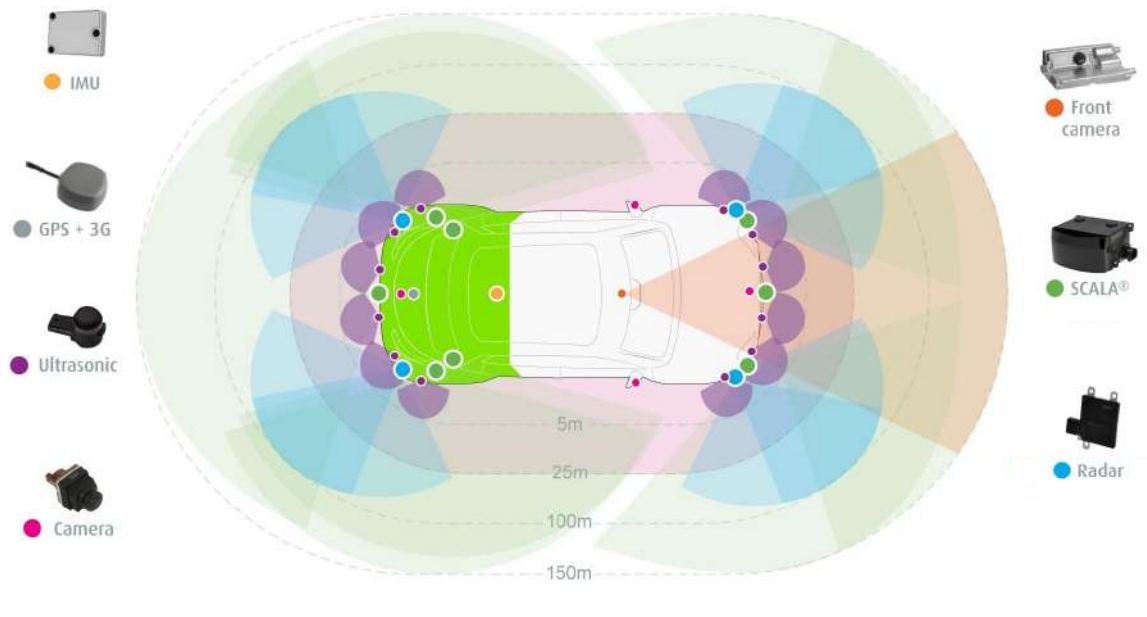


FIGURE 1.3: **Sensor setup of the Valeo Drive4U prototype.** It relies on cost-effective sensors such as cameras, radar and minimal LiDAR for surround perception of the scene; ultrasonic sensors for parking assist and self-parking systems; GPS and IMU for self-positioning relative to the surroundings. Source: Valeo

significantly more expensive, with price tags often exceeding hundreds of thousands of dollars. In contrast, systems aimed at the public market, such as those developed by [Tesla](#), [Wayve](#), and [Valeo](#), employ a more cost-effective approach to autonomous driving. These vehicles mostly rely on less expensive sensors such as cameras, radar and minimal LiDAR (4-beam instead of the expensive 32 or 64-beam) as illustrated in [fig. 1.3](#). An important research challenge is how to achieve the same degree of safety that the over-engineered approach offer, but with cost-effective sensors. This question is at the center of [chapter 3](#). In particular, we address how to combine an extremely sparse 4-beam LiDAR with a camera to get competitive performance with respect to a much denser 64-beam LiDAR.

1.4 AD software stack

The design of the autonomous driving software stack determines the overall performance and reliability of the vehicle. Two popular approaches for developing this software are the modular software stack and the end-to-end deep learning approach. The *modular* AD software stack is a traditional approach that involves breaking down the problem into smaller, manageable components, each of which is tackled by a separate module. Typically, parts of this kind of system rely on expert-knowledge and hard-coded rules. On the other hand, end-to-end deep learning approaches seek to directly map sensor inputs to control outputs (angle of steering wheel and gas pedal), bypassing the need for intermediate representations or modules. This kind of system is entirely learned from data, expert-knowledge only takes the form of inductive bias.

The modular AD software stack is composed of multiple, independent components that work together to enable autonomous driving. It typically follows a “Perceive, Predict, Plan, Act” architecture, see [fig. 1.4](#) for an illustration. The perception module gathers and processes sensory data from camera, radar, and LiDAR sensors. The prediction module takes the outputs from the perception module and predicts the behaviour of other vehicles, pedestrians, and other objects in the driving environment. Based on these predictions, the

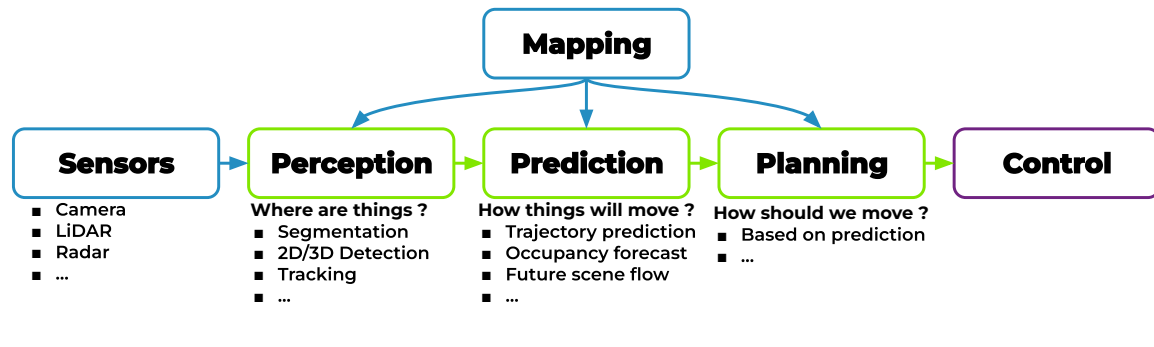


FIGURE 1.4: A schematic illustration showing the classic modular pipeline of autonomous driving.

planning module makes decisions about the ego-vehicle’s trajectory and speed based on its surroundings and objectives (e.g., security, comfort, etc.). At the end of the stack, the control module ensures that the vehicle follows the planned trajectory by controlling the actuators such as the throttle, brake, and steering. Additionally, to integrate prior knowledge, a mapping module can provide pre-recorded information on the environment to each of these modules (e.g., road layout, position of signs and traffic light, etc.) [Liu et al., 2020a]. Although an HD-map greatly simplifies the task of autonomous driving by providing important information on the environment, predicting it from a large quantity of data and updating it over long horizons of time is not trivial. On the other hand, relying on a pre-recorded map constrains the vehicle to operate in a pre-mapped area. Estimating the elements composing an HD-map from the on-board sensors only and in real-time is an active research topic [Casas et al., 2021; Li et al., 2022b].

While this modular perspective offers a certain level of interpretability, the interdependencies between modules can lead to cascading or compounding errors: if a perception algorithm incorrectly identifies a road sign, it can lead to incorrect behaviour from the subsequent planning and control modules. How to model uncertainty at each step of the modular stack such that each module can interpret it and use it to correct its own predictions remains an important challenge for the research community. Furthermore, it can also be more difficult to maintain and update, as changes to one module may affect others, requiring careful coordination and testing to ensure the system continues to function as intended. Likewise, the modular approach can also be less scalable, as the number of modules and their interactions can grow rapidly as the system becomes more complex. Additionally, such approaches may have higher computational overhead, as each module must process its inputs and outputs, potentially leading to slower system performance.

In comparison to the *modular* approaches, the *end-to-end* approaches are about unifying the perception and planning modules, or even up to the control module, into a single, integrated system [Bojarski et al., 2016; Casas et al., 2021; Chitta et al., 2021; Hu et al., 2022a; Kendall et al., 2019]. This may offer several key advantages over the modular approach. Firstly, the end-to-end approach eliminates the need for separate training of different modules; the network is trained as a whole, learning to perform multiple tasks simultaneously. Moreover, it makes for a more efficient and flexible pipeline, as the network can adapt to changing scenarios and additional data without the need to intervene at different levels of the system. It may also alleviate the risk of cascading or compounding errors, as the transformation from raw sensor data to final output is seamless. However, end-to-end deep learning approaches also have their own set of research problems, such as the need for efficient learning techniques to limit the amount of labelled data required to train such systems, or the demand for methods to understand and interpret the internal workings of the neural network [Jacob et al., 2022; Zablocki et al., 2022; Zemni et al., 2023]. Despite these

challenges, the end-to-end approach is becoming increasingly popular in the development of autonomous vehicles.

1.5 Contributions

1.5.1 Outline

After a detailed overview of the different ways deep learning frameworks can be applied to autonomous driving in [chapter 2](#) (the tasks, the datasets, the methods and the challenges), we present the contributions of this thesis.

In this chapter ([chapter 1](#)), we have highlighted a number of research problems crucial for the autonomous driving task. Our focus on the following research questions (RQs) is the core of this thesis:

- RQ1.** How to leverage inexpensive sensors (e.g., camera, minimal 4-beam LiDAR, etc.)?
- RQ2.** How to fuse information from multiple sensors?
- RQ3.** How to alleviate the need for annotated data?
- RQ4.** How to estimate a map of the environment in real time from raw sensors?

- In [chapter 3](#), we present ‘LiDARTouch’ a new method that combines a monocular camera with a minimal 4-beam LiDAR input, typical of laser scanners currently used in the automotive industry. We introduce a new self-supervision scheme (**RQ1**) and study various network architectures to encode this very sparse LiDAR input (**RQ2**). We show that the use of a few-beam LiDAR alleviates critical issues that monocular camera-only methods suffer from; namely scaling ambiguity and infinite depth problems. This work demonstrates that an inexpensive sensor setup (4-beam LiDAR + camera) can reach competitive performances with respect to more costly systems (10× more) relying on 64-beam LiDARs. Moreover, our system, while not requiring any annotation (**RQ3**), also reaches competitive performances with respect to fully supervised approaches that are trained with dense ground-truth depth that are expensive to acquire.
- In [chapter 4](#), we present ‘LaRa’ for Latents and Rays, a general, transformer-based architecture for scene understanding. In this work, LaRa is applied on a car with six cameras to predict binary vehicle segmentation and driveable area segmentation in the Bird’s-Eye-View (BEV) space (**RQ1**, **RQ4**). There are three important parts to the architecture. The first one is, the input composed of *visual* information, essentially feature maps generated by a Convolutional Neural Network (CNN), and *geometric* information, which is a ‘ray embedding’ that encodes the 3D position and orientation of each pixel that gives the network a natural understanding of 3D relationships between camera views. In a second stage, these *visual* and *geometric* information are then compressed into a small collection of latent vectors, acting as an “internal representation” of the scene. Thirdly, this compact, but rich, representation is then re-projected in a space relevant to the end task (discrete BEV space for instance). This work demonstrates that *some* of the geometric and semantic information of a complex scene, captured and aggregated from many sensors, can be efficiently encoded in a very compact, but rich, latent representation.
- In [chapter 5](#), we conclude the thesis by summarizing our main contributions and presenting perspectives for future work.

1.5.2 Publications and implementations

- **Chapter 3** is based on the paper “LiDARTouch: Monocular metric depth estimation with a few-beam LiDAR”, Florent Bartoccioni, Éloi Zablocki, Patrick Pérez, Matthieu Cord, Karteek Alahari, Computer Vision and Image Understanding, *CVIU 2023* ([Bartoccioni et al., 2023]). The code is available at <https://github.com/F-Barto/LiDARTouch>.
- **Chapter 4** is based on the paper “LaRa: Latents and Rays for Multi-Camera Bird’s-Eye-View Semantic Segmentation”, Florent Bartoccioni, Éloi Zablocki, Andrei Bursuc, Patrick Pérez, Matthieu Cord, Karteek Alahari, Conference on Robot Learning, *CoRL 2022* ([Bartoccioni et al., 2022]). The code is available at <https://github.com/valeoai/LaRa>.

The content of this work has been deployed on a prototype autonomous driving system at Valeo with only a change in the learning rate, demonstrating its adaptability and robustness to new conditions.

Chapter 2

Deep learning applied to autonomous driving

Whether it is equipment manufacturers (e.g., [Valeo](#), [Mobileye](#)), existing actors providing robot-taxi services (e.g., [Zoox](#), [Waymo](#), [Cruise](#)) or companies running for high driving automation (e.g., [Wayve](#), [Tesla](#)), actors in the automotive and transport space increasingly make use of deep learning to power their autonomy software stacks. In this chapter, [section 2.1](#) first introduces the tasks commonly found at the heart of autonomous systems and the public datasets that allow us to study them. [Section 2.2](#) details the main challenges deep architectures face in addressing these tasks. Next, [section 2.3](#) delves into how to represent dynamic scenes; that is, how to learn a representation of the world accurate enough to support reasoning, and interaction with the environment.

2.1 Tasks and public datasets

The software that equips autonomous vehicles must enable them to drive safely in complex and dynamic scenes. There are several ways in which this task can be approached and decomposed.

In the typical modular AD stack, the main modules are perception (where are things?), forecast (how things will move?), plan (how should we move?) and act (how do we best follow the plan?). In this thesis, we focus on methods that fall under the perception and prediction umbrella.

First, [section 2.1.1](#) presents the *perception* tasks, which are about embedding all the information necessary to drive including: detecting road users and their interactions; extracting the road geometry (straight, curved, etc.) and its boundaries (drivable area) from physical or semantic delimiters (e.g., curbs or lane markings); detecting and associating all traffic control devices such as signs, lights, and arrows markings to the relevant driving path.

In a second time, the *prediction* tasks, consisting of predicting the future state of the dynamic environment, are presented in [section 2.1.2](#). The borders of the prediction module with the perception and the planning stages can often be blurry. In this regard, [section 2.1.2](#) overviews the various forms that the parameters to predict can take.

2.1.1 Perception

Perception is an essential component of automatized vehicles, enabling them to understand their environment and make informed decisions. This section discusses various perception tasks common in the autonomous driving context to address elementary needs: what are the objects surrounding the ego-car, and where are they? In practice, utilizing a combination of tasks and modalities (LiDAR, images, and other sensors), is necessary for developing safe and reliable autonomous vehicles.

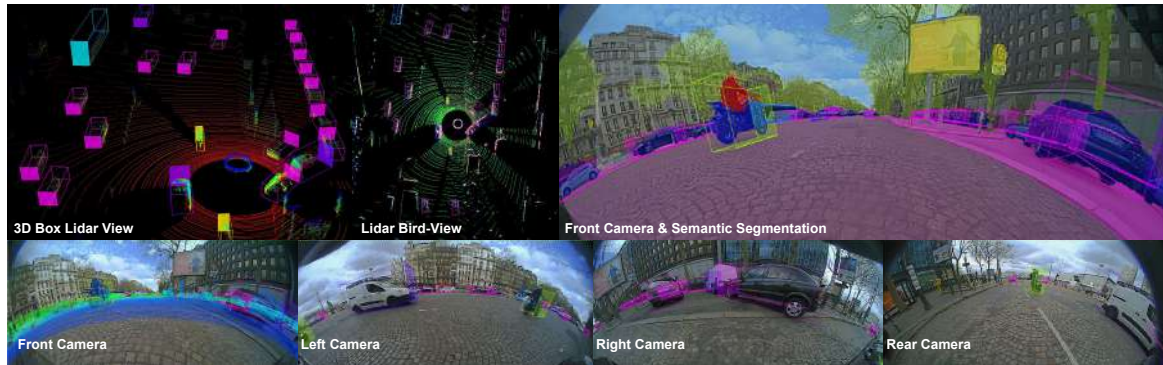


FIGURE 2.1: Examples from the WoodScape dataset [Yogamani et al., 2019] where the vehicle is equipped with four fisheye cameras covering 360° along with a roof-mounted 64-beam LiDAR. Annotations for the tasks of 3D object detection, depth estimation and semantic segmentation are superimposed over the modalities’ visualization.

Where are the objects surrounding the ego-car? First, autonomous systems require an accurate spatial understanding of their surroundings to plan and act safely, and the capacity to estimate *depth* is often used to achieve this [Phillion and Fidler, 2020; Srikanth et al., 2019; Zeng et al., 2019]. Depth estimation is the task of estimating the distance between any element of the scene and the sensor. This information is crucial for determining the relative positions of objects in the scene and understanding the 3D structure of the environment. For such applications, two lines of approach exist to infer depth in a scene, depending on the available data. First, LiDAR-based completion from one or multiple dense LiDARs (e.g., 32 or 64 beams) [Jaritz et al., 2018; Park et al., 2020; Tang et al., 2020; Xu et al., 2019] which uses an additional image signal to increase the number of 3D points in the LiDAR scan of the scene. This is an approach most suitable for typical “over-engineered” Level-5 cars equipped with multiple dense LiDAR (32 beams or more). Secondly, camera-only estimation methods, either stereo [Chang and Chen, 2018; Kendall et al., 2017] or monocular [Casser et al., 2019a; Godard et al., 2019, 2017; Guizilini et al., 2020a,b; Kuznetsov et al., 2017; Mahjourian et al., 2018; Wang et al., 2018; Yin and Shi, 2018; Zhou et al., 2017], estimate the distance to objects from the RGB signal only. This is a more inexpensive approach in terms of sensor cost, but generally less accurate than a LiDAR-based method that provides a physical measurement of the distance.

What are the objects surrounding the ego-car? Knowing the different objects and their positions is crucial for an autonomous vehicle to plan its trajectory and react to its surroundings. This is usually tackled by *semantic segmentation* or *object detection* or a combination of both (e.g., panoptic segmentation systems). These two approaches and their limitations are presented in more detail in section 2.3.1. Semantic segmentation [Phillion and Fidler, 2020; Ronneberger et al., 2015; Vobecky et al., 2022] is the task of classifying each pixel in an image or point in a point cloud into one of several predefined categories, such as “road”, “pedestrian”, “car”, etc. An example of a semantic segmentation mask is illustrated in the top-right image of fig. 2.1. Object detection [Carion et al., 2020; Misra et al., 2021; Redmon et al., 2016; Tan et al., 2020] is the task of identifying and locating objects in the image plane (2D detection) or in the 3D space (3D detection). The goal of object detection is to produce a set of bounding boxes that tightly enclose objects of interest, along with class labels for each object (see fig. 2.1 for an example of 3D detection). While 2D detection only has to fit the pixels of the object in the box, 3D detection needs to infer the entire shape of the object (often only partially visible), its 3D position as well as 3D orientation.

AD-related perception tasks. Other important perception sub-tasks for autonomous driving include lane detection [Chen et al., 2022], which is essential for the vehicle to stay in its lane and navigate safely; traffic signs and traffic lights recognition [Mishra et al., 2022], which is crucial for the vehicle to understand the rules of the road and obey traffic laws; pedestrian detection and behaviour recognition [Belkada et al., 2021; Mordan et al., 2021], crucial to safely navigate in urban areas and cities among vulnerable road users. These tasks, when combined with others, extract very rich information from the environment that are useful for downstream algorithms (prediction, planning and control).

Datasets and benchmarks. Several public datasets are commonly used for training and evaluating these perception tasks, including classic datasets such as Cityscapes [Cordts et al., 2016] and KITTI [Geiger et al., 2012], as well as more recent ones like Argoverse [Chang et al., 2019], WoodScape [Yogamani et al., 2019], nuScenes [Caesar et al., 2020] or Waymo Open [Sun et al., 2020]. These datasets provide high-quality images, LiDAR point clouds, and annotations that can be used to train and evaluate deep learning models for the aforementioned tasks. Figure 2.1 illustrate some of these modalities and annotations present in the WoodScape dataset [Yogamani et al., 2019]. We also give an overview of these datasets in table 2.1.

2.1.2 Motion prediction

In autonomous driving, it is not only necessary to perceive the current state of the environment, but also to anticipate how it will change in the future. That is the role of methods categorized among “prediction” tasks. It allows the vehicle to forecast the future actions of other agents in the scene, such as vehicles and pedestrians, and to anticipate potential hazards. That being said, the forecasting pipeline must overcome numerous challenges, including: modelling the interdependence between agents’ actions in the scene; integrating the constraints imposed by road geometry and traffic rules; modelling the inherent uncertainty in future prediction; handling the partial observability, occlusions and disocclusions of agents.

To this end, deep learning has been leveraged in several stages of the forecasting module, most notably at the input and output levels. In particular, two input representations gained popularity in modern benchmarks for driving scene future prediction [Caesar et al., 2020; Chang et al., 2019; Ettinger et al., 2021]: the vectorized and the rasterized representations. The vectorized approach is essentially a big table where the scene is described by numbers: the lanes are represented as polygons, a set of 3D coordinates; agents and their trajectories as a sequence of 3D coordinates along with values for their attributes such as the size, the category (e.g., car, bus, pedestrian, etc.). A 3D rendering of a vectorial representation is provided in fig. 2.2b. On the other hand, the rasterized approach represents lanes and agents as images from a virtual top-view. Typically, each channel of the image represents an element or a characteristic of an element in the scene. For example, in the work of Bansal et al. [2018], traffic lights are represented by grayscale encoding where each lane center is coloured with the brightest level for red lights, intermediate gray level for yellow lights, and a darker level for green or unknown lights. Also, speed limits are represented with a single channel, where lane centres are coloured in proportion to their known speed limit. Figure 2.2a illustrates how the road layout as well as its rules, namely speed limits, traffic lights, and agents in the scene are represented. Although capable of representing most of the major elements that influence the scene dynamic, these input representations are often computed offline. Usually, the full temporal context (even the future) is available to correctly detect and identify objects in the scene, and manual corrections frequently supplement these offline predictions. While this provides an upper-bound performance

TABLE 2.1: **Overview of autonomous driving datasets.** We present several datasets commonly used to develop and evaluate models for the different tasks discussed in [section 2.1](#). For each dataset, we give the number of samples, where one sample is one recording of the scene from every sensor (e.g., one sample in nuScenes contains a LiDAR scan and the six images from each camera). We also indicate which sensors are fitted on the car, the different tasks for which annotations are available, as well as the visual variations that the dataset contains in addition to the standard "clear sky in daylight".

Dataset	Samples	Sensors	Annotations	night/ rain
KITTI [Geiger et al., 2012]	15k	forward camera LiDAR	Depth Segmentation Detection Flow Road layout	No/No
Cityscapes 3D [Cordts et al., 2016]	20k	forward camera	Depth Segmentation Detection	No/No
Argoverse [Chang et al., 2019]	44k	ring of cameras LiDAR	Depth Segmentation Detection Road layout Ground Height	Yes/Yes
DDAD [Guizilini et al., 2020a]	13k	ring of cameras LiDAR	Depth	No/No
nuScenes [Caesar et al., 2020]	400k	ring of cameras LiDAR radar	Segmentation Detection Road layout	Yes/Yes
Waymo Open [Sun et al., 2020]	230k	front and sides cameras LiDAR	Segmentation Detection Human skeleton	Yes/Yes
ONCE [Mao et al., 2021]	1M	ring of cameras LiDAR	Detection	Yes/Yes
WoodScape [Yogamani et al., 2019]	10k	ring of fisheye cameras LiDAR	Depth Segmentation Detection Sensor soiling	No/No

for prediction algorithms, this perfect representation, devoid of any uncertainty, is a far cry from the online conditions faced by automated vehicles.

At the output level, the most commonly-used forecasting representation on these benchmarks are trajectory sets, which also suffer from severe limitations: their parameterized nature constrains the class of future distributions that can be predicted, and trajectories cannot represent the space that the vehicle will occupy as its shape is not encoded (which is necessary to model shape-shifting vehicles like articulated bus or excavators).

Instead, a general solution to dynamic scene forecast necessitates an encoding of the world, learnable from the raw sensors signals, that is compact and yet rich enough to support reasoning, planning, and interaction with the environment. Ideally, the learned representation should enable inference of current and future states of all key objects in the scene, whether from known categories or from new ones. In short, it requires the representation of the scene to contain all the contextual information necessary for forecasting, directly from sensors. In this line of thought, several works aim at designing and training

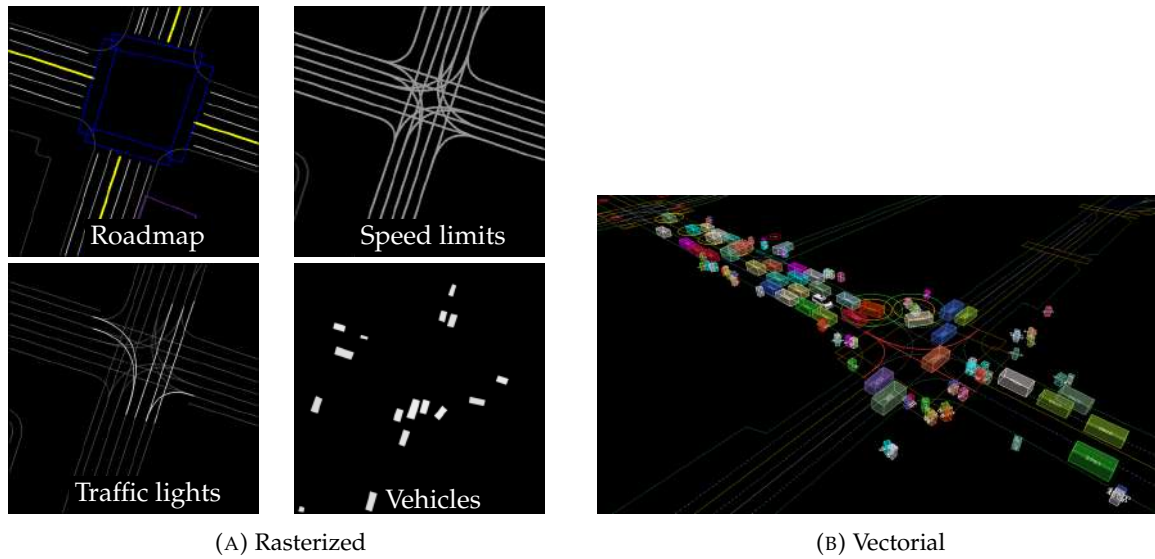


FIGURE 2.2: Illustration of the representations commonly used as input for the motion prediction task as defined on public benchmark [Caesar et al., 2020; Chang et al., 2019; Ettinger et al., 2021]. Here, (B) is a 3D visualization of the vectorial data. Credits to Bansal et al. [2018] and Ettinger et al. [2021].

holistic end-to-end driving systems that ingest raw signals and output either (1) a unified intermediate representation that can fully support planning and control [Casas et al., 2021; Chitta et al., 2021] or (2) directly driving actions [Hu et al., 2022a; Kiran et al., 2020] for AD.

The probabilistic BEV occupancy prediction, close in spirit to rasterized representations, has recently gained a lot of interest as an intermediate representation. BEV occupancy prediction is the task of estimating a probability map that indicates the likelihood of occupancy for each grid cell in the BEV representation. It allows fusing information from multiple modalities into a common representation, naturally handles uncertainty and support planning. More specifically, the future prediction in the BEV space takes the form of a “motion flow” which can directly be used as a cost map by the planning pipeline [Casas et al., 2021; Mahjourian et al., 2022]. More recent datasets for future prediction adopted this representation (e.g., Waymo Open [Mahjourian et al., 2022], nuScenes [Caesar et al., 2020]) and numerous papers were published on how to predict and use it, e.g., [Bansal et al., 2018; Casas et al., 2021; Chitta et al., 2021; Hu et al., 2021; Zeng et al., 2019].

When driving actions are directly estimated, the system is usually based on imitation learning and reinforcement learning. Based on Markov Decision Process (MDP) [Sutton and Barto, 1998], Reinforcement Learning (RL) explicitly models the temporal behaviour of an agent that interacts with a dynamic environment through perception, actions and rewards, and that tries to maximize its expected future cumulative reward (expected “return”). As such, RL mathematically relies on temporal predictions of a completely or partially observed dynamic Markovian model. Short-term forecast, e.g., over one time step of the time-discretized system, is also intrinsically part of the RL model through the MDP’s dynamics, that is the probabilistic distribution over the next state of the agent conditioned on the value of the current state and selected action. Learning this one-step probabilistic prediction function remains an open-problem for driving environments.

2.2 Challenges

While deep learning is a powerful tool, the challenges that come with its use in this AD context are many. The task of creating the driving software stack of an autonomous vehicle is strewn with many challenges, such as the need to learn from data with few or no annotations, or being robust to faulty sensors and visual impediments. This section delves into some of these challenges and explores the various approaches proposed to overcome them.

2.2.1 Modalities of very different natures

When it comes to applying deep learning to autonomous driving, one of the biggest challenges is dealing with inputs of very different natures. For example, one might have to process an image from a camera, a point cloud from a LiDAR, or energy spectrums from a radar. Hence, different architectural specificities are needed to handle these types of modalities.

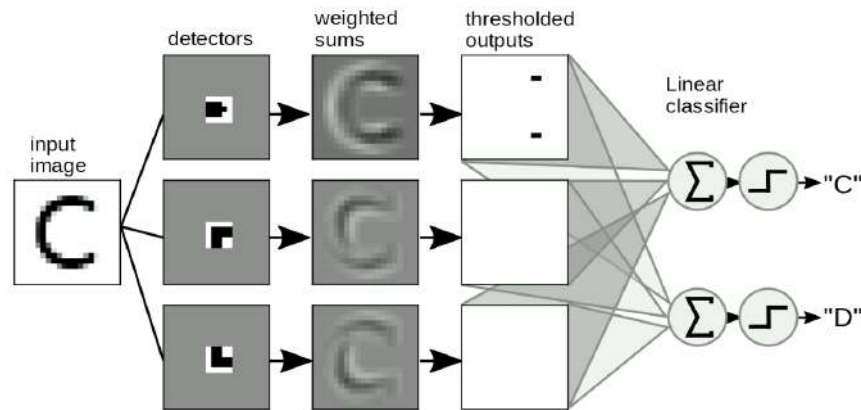


FIGURE 2.3: Illustration of a simple CNN-based classifier.¹ Here, a bank of simple pattern detectors are applied to an image of a “C” letter and produce a collection of feature maps. Next, the linear classifier aggregates information from each detector output and makes its prediction. The detector at the top row, representing an extremity, is the most discriminative feature between the letter “C” and “D”, hence the most “active” detector for the image.

Processing images. The most popular method to process images from cameras are CNNs. A CNN is formed by successive “layers” of artificial neurons, each layer transforming its input via a collection of localized linear 2D filters followed by piece-wise non-linearities, also called “activation functions”. Each filter can be seen as a *trained* local pattern detector. Consequently, each layer ingests a stack of 2D detected features (“feature maps”) and outputs another one (possibly of different dimensions and a different number of features). The deeper in the network, the larger is the spatial extent (“receptive field”) of the produced pattern detector. These detectors usually outperform hand-crafted feature extractors. These networks can extract features from images that, further down, enable the identification of objects and other relevant information. CNNs can be supplemented with a final “decision” layer (e.g., a linear projection with a logistic function) to create a standalone network (e.g., a “classifier”) or used as a “backbone”; that is, a CNN producing high-level features for downstream blocks of the deep architecture. Figure 2.3 illustrates a

¹Courtesy of <https://atcold.github.io/pytorch-Deep-Learning/fr/week03/03-1/> - [Last accessed on 2023-01-19]

simple 1-layer CNN-based classifier. In practice, many more layers are used. Specifically, CNNs such as ResNet [He et al., 2016], EfficientNet [Tan and Le, 2019], or ConvNeXt [Liu et al., 2022b], are very popular backbones in deep architectures and typically use between 15 to 100s of layers. To predict in the image space, a common architecture type is the fully convolutional network in an encoder-decoder fashion, that is, a network only composed of convolutional layers. The encoder, generally built on top of a backbone network, produces feature maps at multiple levels of resolution from the input image. These feature maps are then fed into the decoder, a series of convolutional and upsampling layers. This produces a pyramid of feature maps at multiple scales, which enables the retention of fine details or detection of objects of various sizes. In particular, the U-Net [Ronneberger et al., 2015] (initially developed for biomedical image segmentation) and Feature Pyramid Networks (FPNs) [Lin et al., 2017; Tan et al., 2020] (typically used in detection pipelines) are among the most used fully convolutional architectures (see fig. 2.4 for illustrations).

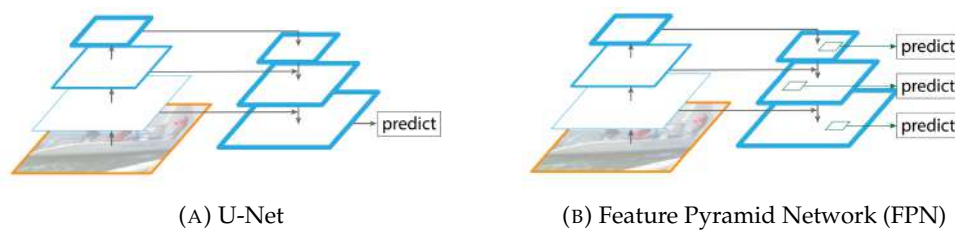


FIGURE 2.4: Illustration of the fully convolutional networks U-Net [Ronneberger et al., 2015] and FPN [Lin et al., 2017]. Most notably, the U-Net makes the prediction at the highest level, while the FPN makes a prediction at each scale, which are then combined.

Processing point clouds. On the other hand, the methods to process point clouds are more diverse and can be divided into three main categories: (1) multi-view (2) volumetric and (3) raw or point-based methods (see fig. 2.5 for a representation of each). Multi-view based methods transform the point cloud into 2D images by projecting the point cloud into multiple virtual views [Su et al., 2015], or the camera plane [Chen et al., 2017] (when one is available), a range image [Biasutti et al., 2019] (spherical projection of a LiDAR point cloud on its intrinsic 2D lattice) or BEV map. Then, the point cloud now encoded as an image of some form, a traditional 2D CNN can be applied to process the signal. Volumetric-based methods discretize the continuous 3D space into a volumetric grid and then process it by 3D convolutions [Maturana and Scherer, 2015]. Such approaches, constrained by the volumetric resolution and the computational cost of 3D convolutions, need sparsity optimization like octrees [Riegler et al., 2017] or sparse convolutions [Choy et al., 2019]. On the other hand, point-based methods [Boulch et al., 2020; Qi et al., 2017; Thomas et al., 2019], allow the network to maintain the fine-grained structure of the point clouds. It is to be noted that these three representations are not exclusive and can be combined [Xu et al., 2021].

Transformers as a general architecture. In contrast to these modality-specific architectures, often requiring domain expertise, the Transformer is a very general type of architecture [Vaswani et al., 2017]. The Transformer architecture was initially developed to address the limitations of previous deep learning models for natural language processing tasks. In particular, early models relied on recurrent or convolutional layers which have difficulties capturing longer-range dependencies, making it difficult to process long sequences of data, such as a book. The Transformer architecture can extract complex dependencies in long sequences of data thanks to its self-attention mechanism, which allows the model to directly

²https://github.com/risteon/blender_kitti - [Last accessed on 2023-01-19]

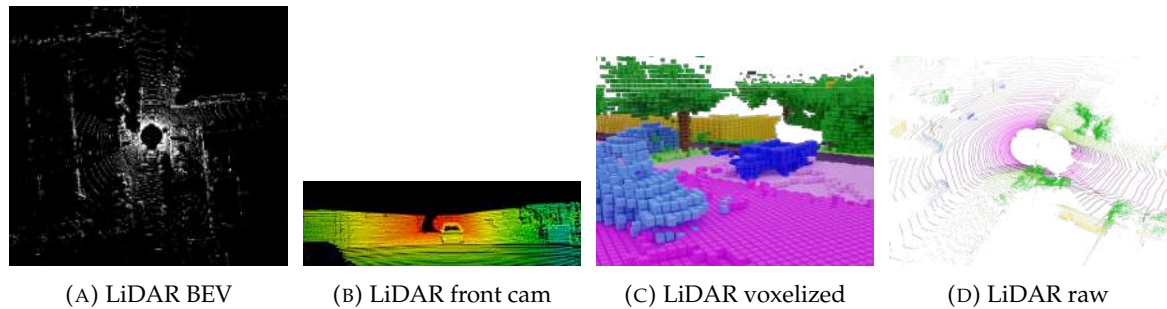


FIGURE 2.5: **Representations of point clouds.** Courtesy of (A) [Chen et al. \[2017\]](#) (B) [Qi et al. \[2018\]](#), (C) and (D) blender-kitti.²Note that (C) and (D) are coloured with semantic classes.

attend to any part of the input sequence, regardless of its position in the sequence. The self-attention layer ([fig. 2.6](#)) takes as input a set of vectors and each vector is transformed into three separate vectors: a query, a key, and a value (“QKV triplet”). The main idea is to create a learnable, high-dimensional, indexing system from the input vectors where the queries ask “this is the information I am looking for”, the keys “this is the information available” while the value vector represents the actual information. More formally, the query and key vectors are used to compute an “attention score”, which is a measure of how much the query vector “matches” the key vector. When computed over the full set of input vectors, it creates an “attention map” that indicates the importance of each input vector with respect to the others. The final output is computed as a weighted sum over value vectors using the attention map weights.

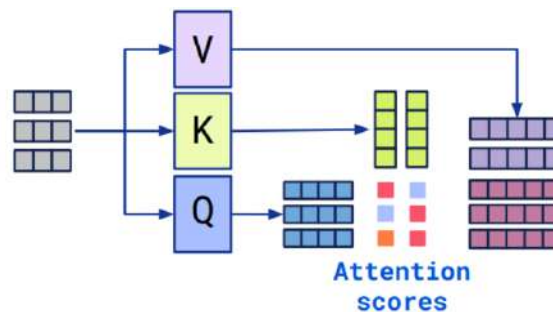


FIGURE 2.6: Schematic illustration of the self-attention layer [[Vaswani et al., 2017](#)]. The input (in gray) is transformed into query (Q), key (K), and value (V) vectors which are used to model complex dependencies thanks to an attention system and form the output (in dark pink). Adapted from [[Jaegle et al., 2022](#)]

Tokenization and positional embeddings. One of the main advantage of Transformers is that their only requirement is for the input to be organized as a set of vectors, called ‘tokens’. For example, in vision transformers [[Dosovitskiy et al., 2021](#); [Liu et al., 2021](#)], images are divided in small patches of pixels (e.g., 16×16 in [[Dosovitskiy et al., 2021](#)]) that are flattened (i.e., reorganized to be 1D) and embedded by a linear projection. These ‘patch embeddings’ are the tokens given to the transformer. Point clouds can also be tokenized, either in their raw form where each point is considered a token or by first voxelizing it (i.e., akin to 2D patches but in 3D) [[Lu et al., 2022](#)].

Another important concept used in transformers is the positional embedding. When combined with the input signal (typically by addition or concatenation), it is a way to

introduce an *inductive bias* and help the network focus on extracting specific dependencies over others. These embeddings can take various forms and are defined according to the properties of the input signal. For example, if the input is a multi-dimensional grid (e.g., an image, a video, a voxelized volume) spatial knowledge can be introduced by augmenting the input tokens with an embedding of axis coordinates (x and y coordinates for an image) [Jaegle et al., 2022]. This can help the network extract better spatial relationships [Bartoccioni et al., 2022; Guizilini et al., 2022b; Jaegle et al., 2022]. For multi-modal setups, an embedding for each modality can be considered, helping the network to better identify features from each modality leading to better fusion of information [Jaegle et al., 2022]. Moreover, when dealing with multiple views, an embedding of the cameras' parameters, notably their positions and orientations, is sufficient for the network to extract complex correspondences between views. For instance, Yifan et al. [2022] and Guizilini et al. [2022b] used this principle for multi-view stereo depth estimation without expensive matching volumes [Kendall et al., 2017]. In particular, we show in [chapter 4](#) that augmenting each pixel with their ray embedding gives the network a natural understanding of 3D relationships between views.

2.2.2 Robustness to faulty sensor and visual impediments

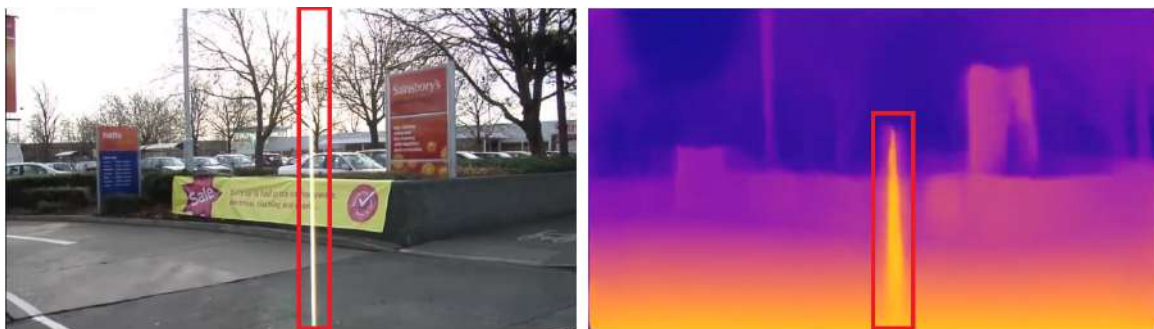


FIGURE 2.7: Depth estimation artefact from a glare, highlighted in red, on an example from the KITTI dataset [Geiger et al., 2012]. Depth map predicted using the method from Godard et al. [2017].

Sensors can be faulty or obstructed, leading to inaccurate or missing information. One of the biggest challenges in autonomous driving is ensuring robustness to these issues. For example, cameras have limitations such as sensitivity to lighting conditions, which can make it difficult for the network to accurately identify objects in nighttime condition or in the presence of glares (see [fig. 2.7](#)).

In particular, sensors directly exposed to the external environment are likely to be soiled. One way to deal with such a case is to use a soiling detection system [Uricar et al., 2019] combined with a physical actuator (e.g., a jet of water to remove dirt on the sensor).

Alternatively, sensor fusion is a common approach to mitigate these issues using the principle of redundancy. This involves combining data from multiple sensors to create a more accurate and reliable representation of the environment. Ways to fuse data include early and late fusion schemes. In early fusion, the data is combined before being processed; the different modalities are typically concatenated at the very beginning of the deep architecture. For the late fusion approach, the data is first processed separately and then combined at the very end (e.g., by multiplying the probabilities from both streams). More recently, deep multi-stage fusion has become a favoured approach with deep learning architectures. This can be implemented in different ways: projecting the data of multiple sensors into a single intermediate representation, allowing for more accurate and efficient

sensor fusion [Harley et al., 2022; Hendy et al., 2020] or directly fusing information from intermediate feature maps between both modalities [Chitta et al., 2022; Li et al., 2022a; Zhao et al., 2021].

2.2.3 Training with few or no annotations

Collecting and annotating large amounts of data is a crucial aspect of training deep learning models for autonomous driving. However, it is time-consuming and costly to collect and annotate enough data to train a model for a specific task or context. For example, the depth ground-truth in the KITTI dataset [Geiger et al., 2012] required a \$60k LiDAR combined with a stereo camera.

One very common approach to alleviate this problem is *transfer learning*; using a pre-trained model on a related task or dataset and fine-tuning it for the specific task or context. Often, CNNs are pre-trained for the classification task on ImageNet [Deng et al., 2009] while pre-training for 2D detection seems to be a promising avenue for BEV predictions [Wang et al., 2022].

An alternative is to leverage the large amounts of data that cars collect while driving using *self-supervised learning*. The idea behind self-supervised learning is to learn from the structure of the data itself, rather than relying on explicit annotations. Typically, the model is supervised on a pretext task that does not require manual labelling. The aim is to learn representations that transfer well to downstream tasks of interest. A representation that “transfers well” is one that significantly reduces the amount of annotation required to reach a set level of performance. Most works in self-supervised learning for vision [Caron et al., 2021; Gidaris et al., 2021; Grill et al., 2020; He et al., 2022; Henaff et al., 2021; Komodakis and Gidaris, 2018] were developed to learn on still images in well-defined datasets: only one or few objects are centred in the image; the distribution of classes is close to uniform; images are small. In these conditions, simple augmentations (e.g., random cropping, affine transformation, colour jittering) are enough to learn a representation that “transfers well”. However, the performance of these methods degrades when applied directly on AD datasets [Chen et al., 2021].

In the context of *self-supervised learning for AD*, popular frameworks include exploiting the fact that the car moves, using geometric principles as supervisory signals and leveraging other modalities than images. For example, FlowE [Xiong et al., 2021] builds on top of BYOL [Grill et al., 2020] but instead of a contrastive objective at the image level they train on a pixel-wise objective that makes features equivariant to optical flow (i.e., an object is still the same object even if it moves). This representation, when fine-tuned for semantic segmentation, outperforms the fully-supervised model with only 10% of the labelled data. Another line of work built on insights from predictive coding theories [Harley et al., 2019; Lal et al., 2021], where the model is trained to predict what a scene would look like from different viewpoints (e.g., the next few frames). This objective constructs a representation that is able to “imagine” occluded information (“amodal completion”), track objects over time (features change smoothly over time), and improve 3D object detection.

Apart from pre-training tasks, even the downstream task can be self-supervised. For instance, self-supervised depth estimation from monocular cameras recently became very popular [Casser et al., 2019a; Godard et al., 2019, 2017; Guizilini et al., 2020a]. Leveraging a set of consecutive frames, this paradigm predicts neighbour views by means of view projection using predicted dense depth maps and the relative changes in pose. The model is trained by minimizing a photometric reconstruction; a view well predicted entails a correct estimation of the depth map and poses. These approaches are discussed in more detail in the following sections, and also in [chapter 3](#).

Some works also exploit the synchronization between LiDAR and cameras on typical AD vehicles. In particular, [Vobecky et al. \[2022\]](#) use synchronized images and LiDAR point clouds to generate pseudo-groundtruth for semantic segmentation learning without manual annotations. Also based on LiDAR-cameras synchronization, the work of [Sautier et al. \[2022\]](#) distils self-supervised pre-trained image representations into 3D models. This allows pre-training a model operating on 3D point clouds that transfer well on semantic segmentation and object detection tasks. Knowing the calibration of each sensor, correspondences can be established between pixels and 3D points, and thus, a contrastive objective to impose 3D point features and 2D pixel features to match; infusing semantic knowledge in the 3D features.

2.2.4 Transformation of representation

As discussed, in the context of AD, predictions can be diverse: e.g., 2D or 3D boxes, segmentation masks, BEV map, 3D voxels, or even graphs to represent the pedestrians' skeleton. This poses the need to transform between representations. Cameras being the most frequent sensor fitted on cars, these predictions are often made from camera images only. Predicting 3D voxels or 3D skeletons for pedestrians from pixels only is far from being trivial and may require to make additional estimations (e.g., depth, 2D detection, etc.), making the system prone to compounding errors.

In this aspect, Transformer architectures [[Vaswani et al., 2017](#)] are, again, becoming very popular because of their generality and impressive performance, and are now a preferred choice to infuse information from one representation to another. Similar to self-attention, the cross-attention mechanism [[Vaswani et al., 2017](#)] is at the core of this wide adoption. While self-attention is a mechanism that allows the model to focus on specific parts of a single input sequence of data, cross-attention, on the other hand, allows the model to focus on the relationships between *different* sequences of input data. For example, cross-attention can be used to establish relationships between pixels in the camera images and pixels in a Bird's-Eye-View image. This can be used to project image features into BEV features [[Chitta et al., 2021](#); [Zhou and Krähenbühl, 2022](#)] without the need for dense depth estimation. The same scheme has been used for: 2D or 3D supervised object detection that does not need complex hand-crafted detection pipelines [[Carion et al., 2020](#); [Misra et al., 2021](#)]; learning object-centric representations in a self-supervised way, obtaining segmentation masks without explicit supervision for segmentation [[Locatello et al., 2020](#)]

A Transformer-based architecture that has recently gained popularity in the field of deep learning is the Perceiver [[Jaegle et al., 2021](#)] and its extension, the Perceiver-IO [[Jaegle et al., 2022](#)]. The basis of the architecture (illustrated in [fig. 2.8](#)) is to first project, using a cross-attention, the input tokens into a small collection of latent vectors. This set of latent vectors is typically much smaller than the inputs, which makes it cheap to process. To make the final prediction, the latent representation is then re-projected into the space of output. Such architectures excel at extracting complex and long-range dependencies, even from very large inputs and outputs. For example, it allows for optical flow and multi-view stereo depth estimation without the need for expensive cost volumes aggregating the matching costs over different possible disparities, usually required to find correspondences. In contrast, all the computation and matchings are done in this intermediate abstract space [[Guizilini et al., 2022b](#); [Jaegle et al., 2022](#)].

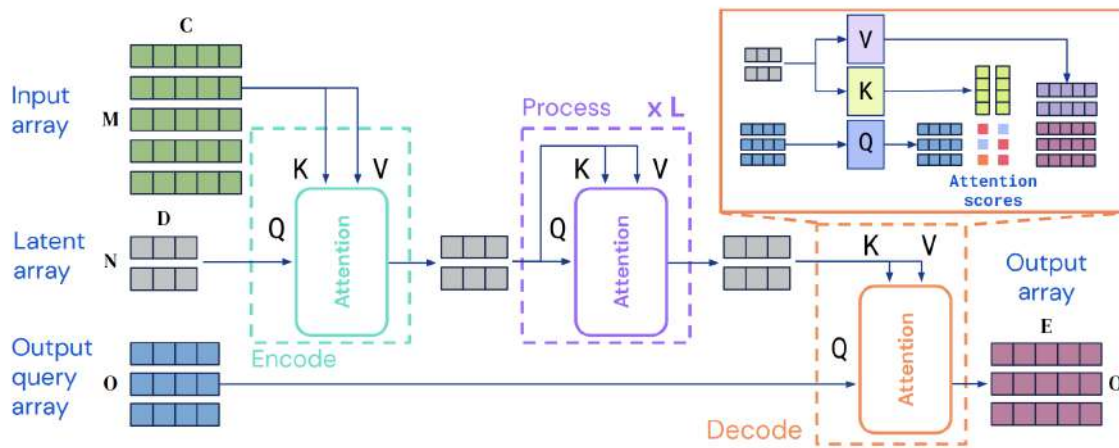


FIGURE 2.8: Illustration of the Perceiver IO architecture [Jaegle et al., 2022]. A first cross-attention layer projects the input vectors into a small collection of learnable, abstract, latent vectors. The final prediction is obtained by a re-projection of the latent representation into the output space. There are two main advantages to this type of architecture: (1) the size of the internal representation is decoupled from the inputs and outputs, hence information can be extracted from this compressed space in a very efficient manner, (2) the input and output spaces can be of very different nature, the latent array will be the “bridge” between the two.

2.3 Representing a dynamic scene

This section provides an overview of different deep-learning based approaches to represent dynamic scene in the context of autonomous driving, namely: in the image plane (2D), as a 3D point cloud or as a set of 3D boxes, in the Bird’s-Eye-View plane, or in a high-dimensional implicit representation. Dynamic scenes present a unique challenge in the context of autonomous driving due to a constantly changing environment and conditions that vehicles must operate in (e.g., traffic, road infrastructure, weather changes). Unlike static scenes, dynamic scenes are composed of many moving elements such as vehicles, pedestrians, and other obstacles, which can interact with each other in complex and uncertain ways. This constant motion and unpredictability can make it difficult for autonomous driving systems to accurately perceive and predict the behaviour of these elements, leading to potential safety hazards. The ability to accurately represent and understand a dynamic scene is crucial for the development of safe and reliable autonomous vehicles.

2.3.1 In the camera image plane

One of the main approaches taken by the computer vision community to represent dynamic scenes has been to use estimations in the 2D image plane. Such a system is usually conceptualized in three parts, namely the ontological, geometric and dynamic aspects.

Ontology (semantic segmentation and detection). The ontological aspect relates to the conceptualization of things composing a dynamic scene; in the words of Gruber [1995], it is “an abstract and simplified view of the world we want to represent”. Broadly speaking, an ontology is defined using domain-expert knowledge and entirely defines what are the passive and active agents, their functional attributes, their relationships and possible interactions. Specifically, semantic segmentation [Ronneberger et al., 2015; Vobecky et al., 2022] and object detection [Carion et al., 2020; Redmon et al., 2016; Tan et al., 2020] are ontological tools that make it possible to represent domain knowledge in a form that can be used by a

machine. While being powerful tools, they still face many challenges. First, the ontology defined by the expert must be complete, it must encode everything that is needed to drive, i.e., what is not described in the ontology does not exist. The world we live in changes continually, and it must be possible to add new concepts to the ontology without having to modify its foundations. Moreover, the ontology expressed in the 2D plane of the camera image does not provide a full understanding of the 3D space.

Geometry (depth estimation). A spatial understanding of the surroundings is required to plan and act safely, and the capacity to estimate *depth* is often central to achieving this [Philion and Fidler, 2020; Srikanth et al., 2019; Zeng et al., 2019]. For such applications, two lines of approach exist to infer depth in a scene: LiDAR-based completion and camera-only estimation methods. LiDAR-based depth *completion* methods produce depth maps from one or multiple *dense* LiDARs (e.g., 32 or 64 beams) [Jaritz et al., 2018; Park et al., 2020; Tang et al., 2020; Xu et al., 2019] and essentially interpolate the scene structure from the input signal. However, these approaches rely on expensive setups, and often require several steps of post-processing to produce the final supervisory signal [Geiger et al., 2012], making the cost for annotation acquisition very high. In recent years, there has been an increased interest in exploring methods for self-supervised, camera-only, depth estimation [Casser et al., 2019a; Godard et al., 2019, 2017; Guizilini et al., 2020a; Mahjourian et al., 2018; Wang et al., 2018; Zhou et al., 2017]. Self-supervised learning methods, leveraging widely available, low-priced sensors, and not requiring any human annotation, are particularly scalable with the data acquired by a vehicle fleet. In this thesis, self-supervised depth estimation from a monocular camera is at the core of our work in [chapter 3](#). The central idea of such approaches is to combine pose and depth predictions to project a neighbouring source image into the target view. The objective is based on photometric reconstruction, a surrogate task aimed at resynthesizing a target image, given neighbouring source images with different viewpoints [Godard et al., 2019; Ma et al., 2019; Zhou et al., 2017]. The underlying intuition is that to correctly resynthesize the target view from the source one, both the depth and pose estimation must be accurate. An overview of this learning process is illustrated in [fig. 2.9](#).

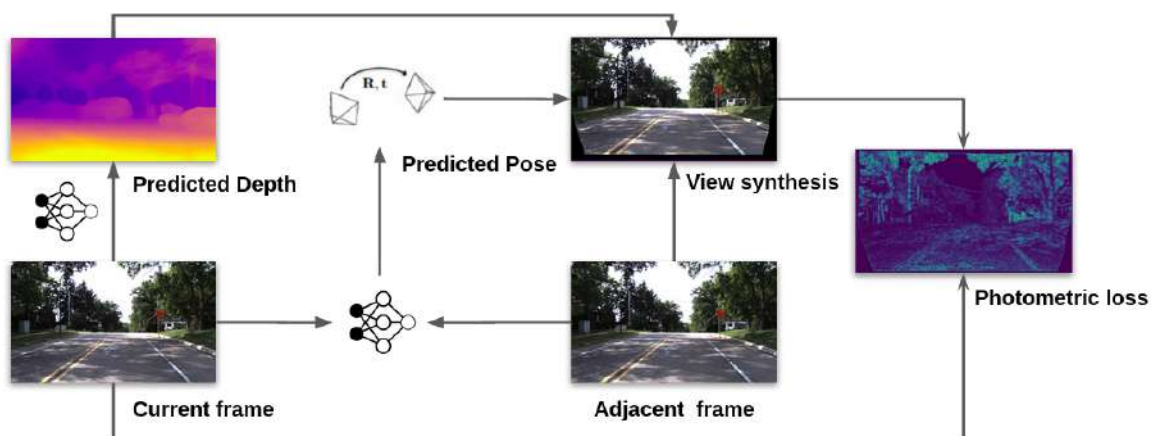


FIGURE 2.9: Illustration of a self-supervised learning system for depth estimation.³ A depth and a pose networks are used to synthesize the current frame from a temporally adjacent frame. The photometric loss between the original and synthesized images is minimized during training. This approach is driven by the intuition that to correctly resynthesize the target view, both the depth and pose estimation must be accurate.

³Adapted from <https://medium.com/toyotaresearch/self-supervised-learning-in-depth-part-1-of-2-74825baaa04>

Dynamic (optical flow). Identifying moving objects in the scene can provide critical information about the environment, such as the presence of other vehicles, pedestrians, and bicycles. Objects that are in motion are more likely to be hazards, as they may be unpredictable and difficult to avoid. In the context of 2D representation, movement is often represented using *optical flow*, that is, the task of determining the motion parameters of pixels (or moving parts) between consecutive frames in a video [Jaegle et al., 2022; Xiong et al., 2021].

Nonetheless, prediction in the image plane like depth and optical flow do not come without their limitations. For example, cameras have a limited field of view and can only capture a small portion of the environment at any given time, such 2D outputs do not allow representing or imagining what is not seen. Likewise, objects or obstacles can be occluded, which can make it difficult to accurately identify and track them from a 2D representation only.

2.3.2 In the 3D space

Predictions can also be made in the 3D space directly. Point clouds are well-suited for representing dynamic scenes as they provide a rich representation of the 3D geometry of the environment, including the shape and location of objects. To represent dynamic scenes, a series of point clouds captured over time can be used [Wang and Tian, 2022]. Specifically, motion information, named 3D ‘scene flow’ [Jund et al., 2022], is similar to 2D optical flow and can be estimated based on consecutive point cloud frames. Also, by analysing the changes in the point cloud over time, it is possible to identify and track objects in the environment. While this is typically done from LiDAR inputs [Wang and Tian, 2022], a 2D camera image can be back-projected using the camera parameters to obtain a 3D point cloud that can also feed into LiDAR-based 3D detection systems [Simonelli et al., 2021; Wang et al., 2019], resulting in a ‘pseudo-LiDAR’ system. Likewise, Yang and Ramanan [2020] combine optical flow and motion-in-depth (ratio between the depth of corresponding points over two frames) to recover the 3D scene flow from monocular cameras only.

Extracted cues such as 3D bounding box predictions for objects in the scene as well as road layout in the form of vectors can be used to represent dynamic scenes. In fact, many forecasting methods favoured this representation as it abstracts all the visual variations while keeping the majority of the information needed to predict the evolution of agents in the scene [Bansal et al., 2018; Gao et al., 2020; Nayakanti et al., 2022; Zhao et al., 2020]. However, such detection-based perception systems often involve manually tuned thresholding scores to trade off precision and recall, which may cause the loss of critical information (e.g., an object on the road).

2.3.3 In the Bird’s-Eye-View

An alternative to detection-based representation are probabilistic occupancy grids. Probabilistic Bird’s-Eye-View (BEV) occupancy prediction is the task of estimating a probability map that indicates the likelihood of occupancy for each grid cell in the BEV representation. As already stated in section 2.1.2, the BEV representational space, a.k.a. top-view occupancy grid, recently gained considerable interest within the community for downstream driving tasks, including motion forecasting [Casas et al., 2021; Hu et al., 2021; Mahjourian et al., 2022] and planning [Caesar et al., 2021; Casas et al., 2021; Chitta et al., 2021; Zeng et al., 2019]. The BEV representation is also at the centre of chapter 4, which describes how to estimate it from cameras only.

The increased interest in the BEV grid representation certainly comes from the many advantages it offers. First, BEV appears as a suitable and natural space to fuse multiple views [Hu et al., 2021; Phillion and Fidler, 2020] or sensor modalities (e.g., camera images and LiDAR point clouds) [Bai et al., 2022; Hendy et al., 2020]. Knowing the calibration parameters of each sensor, their respective signal can be projected into the BEV space, then acting as a common space in which modalities of very different nature can share information — an important aspect for sensor fusion (section 2.2.2). Indeed, LiDAR point clouds are natural 3D signals, making their projection into a top-down grid trivial. For camera signals, there are different strategies for this image-to-BEV transformation. These include methods that make this transformation by means of geometric projections [Ng et al., 2020; Phillion and Fidler, 2020; Sengupta et al., 2012; Srikanth et al., 2019] (e.g., by predicting the depth for each pixel and using cameras’ parameters) and other that learn it [Pan et al., 2020; Roddick and Cipolla, 2020; Zhou and Krähenbühl, 2022], e.g., by direct correspondences between all pixels in camera images and all pixels in the BEV grid. These will be detailed in chapter 4.

The BEV representation also has a geometric meaning: it is a discretization of the 3D space where the vertical dimension has been flattened. The compression of the vertical axis is possible as most of the dynamic in a driving scene happens at the ground level, i.e., the plane of the road. This geometrical grounding makes the BEV grid suitable to describe the semantic, geometric, and dynamic aspects of objects in the environment.

The BEV representation can describe a wide variety of objects as well as their future occupancy. The future occupancy takes the form of a “motion flow” where the dynamic of each pixel composing an object is described [Casas et al., 2021; Mahjourian et al., 2022]. This allows modelling complex dynamic events like a trailer swinging uncontrollably behind a car. Otherwise, the prediction can simply take the form of a sequence of occupancy grid, i.e., without associations between pixels over time (“flow”). In all cases, this probabilistic map can be used directly as a cost map by the planning pipelines, where prospective driving paths are weighted by the occupancy likelihood predicted on the pixels they cross [Casas et al., 2021].

2.3.4 Implicit

An implicit representation encodes the state of the world in an abstract, high-dimensional space. In the autonomous driving context, it embeds in an abstract space all the information required for driving such as the geometry, the position, the dynamic and the attributes of objects in the scene, without explicitly representing these attributes (e.g., MILE from Hu et al. [2022a]). In comparison, most of the approaches discussed so far try to map sensory information to an explicit 2D or 3D Euclidean space (whether discrete or continuous). This has limitations that can make such representation insufficient for capturing the dynamic and complex nature of the environment in autonomous driving.

For example, grid-based methods like BEV occupancy grids (section 2.3.3) or 3D voxels (section 2.3.2), essentially learn a lookup table of the scene where every spatial coordinate is mapped to sensory information. This can suffer from discretization errors and limited resolution, making it difficult to accurately identify objects or features in the environment. Figure 2.10 illustrates how pedestrian occupancy is hard to estimate for current methods. Additionally, although mostly storing information about empty space, reasoning in this representation is both memory and computationally expensive because every location must be visited by the algorithm (usually with several layers of convolution).

Methods in the 2D camera plane (section 2.3.1) or in 3D LIDAR point clouds (section 2.3.2) try to make predictions for every sensory element (pixel or 3D point). Both

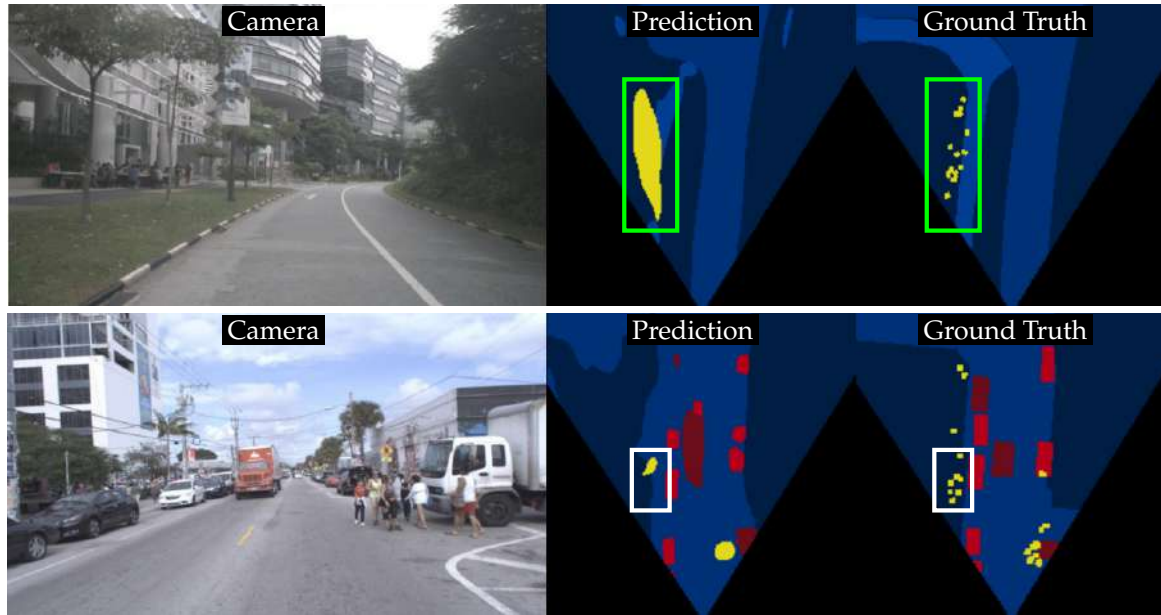


FIGURE 2.10: Columns, from left to right, represent the input image from the frontal camera, the BEV semantic segmentation prediction from a state-of-the-art architecture [Can et al., 2022], and the ground truth. These examples illustrate the resolution problem that the Bird’s-Eye-View representation faces. The coloured boxes (green for the first example and white for the second) highlights the difficulty that a network has to segment “small” objects, pedestrians in this case. This figure is adapted from [Can et al., 2022]

pixel-wise or point-wise predictions require a high level of detail to represent all the objects of the environment (e.g., cones, children, potholes), which can be computationally expensive.

Vectorial representations (section 2.3.4) essentially ask for perfect detection predictions while being limited by their ontology (which currently only integrates cars, pedestrians, and some animals in mainstream datasets [Caesar et al., 2020; Sun et al., 2020]).

In contrast, the implicit representation aggregates the information from every sensor into a common encoding of the environment in a compact, abstract, and efficient manner [Bartoccioni et al., 2022; Guizilini et al., 2022b; Hu et al., 2022a; Sajjadi et al., 2022]. Typically instantiated as a single or a collection of high-dimensional vectors, its compact nature discards irrelevant details and helps efficiently capture complex dependencies. For example, a good implicit representation for dynamic scenes may encode the position, movement, and shape of agents in the scene while ignoring the colour of the sky or the movements of leaves on trees. Also, the representation being abstract and continuous, it alleviates any resolution issue that grid-based systems face while minimizing the memory cost for the scene encoding. Additionally, with irrelevant details being removed, it becomes much more computationally efficient for the neural network to extract and reason about relationships between parts of the scenes (e.g., associating a traffic light to the road it controls, taking into account the distance between a crosswalk and a pedestrian or the interactions between drivers). Overall, an implicit representation is all about managing the trade-off between computational cost and representation quality.

Once learned, such an encoding of the world can be used in various ways. Its aggregative and implicit nature makes it suited for multi-sensor setups and adaptable to a wide range of tasks, an aspect at the core of chapter 4. Most often, it feeds into an RL-based

driving system, where driving commands (angle of the steering wheel, braking, and gas pedal) are directly predicted from the representation [Hu et al., 2022a]. Such a representation, being compact and mostly filled with relevant information, makes it easier to reason temporally and to predict the future state of the world (akin to *world-models* [Bryson et al., 1979; Ha and Schmidhuber, 2018; Hafner et al., 2023; Hu et al., 2022a]), a critical characteristic to enable end-to-end driving.

2.4 Positioning

In this section, we reviewed the existing literature on scene understanding systems for autonomous driving and identified the challenges associated with their creation. In view of these challenges and the research questions established in [chapter 1](#), we now give a high level perspective on the methods we develop in this thesis. For instance, we propose two novel methods that leverage automotive-grade sensors for scene understanding.

In [chapter 3](#), we present a new self-supervised learning system, leveraging a few-beam LiDAR and a camera, for the task of depth estimation. We highlighted in [section 2.1](#) that depth prediction methods are often used to achieve a 3D perception of the surroundings in camera-based autonomous systems. The geometry of the scene can be physically measured from multiple LiDAR sensors, but this requires an expensive sensor setup. On the other hand, camera-only methods are more cost-effective but suffer from two major problems: scale ambiguity and infinite-depth. In particular, the infinite-depth problem results in the depth of moving objects being dangerously overestimated. This makes existing methods unsafe to deploy in real-world scenarios, where moving objects are common. Despite this, the infinite-depth problem is often overlooked and not properly evaluated. To address these limitations, we propose to leverage widely available automotive-grade sensors and combine a monocular camera with a few-beam LiDAR. More specifically, we build on previous self-supervised camera-only methods relying on view reconstruction principles for the task of depth estimation ([section 2.3.1](#)). However, we complement the classic, ill-posed, reconstruction objective with the help of “touches” from a few-beam LiDAR to disambiguate the estimation of moving objects. We validate the influence of this sparse LiDAR integration at different levels of the self-supervised learning scheme. In addition, we introduce a new metric to quantitatively measure this infinite depth phenomenon and highlight its presence in prior methods.

In [chapter 4](#), we propose a transformer-based model for vehicle and driveable area segmentation in the Bird’s-Eye-View (BEV) from multiple cameras. [Section 2.1.2](#) and [section 2.3.2](#) highlight the issues with detection-based perception systems and trajectory-based prediction systems, namely the difficulty to model spatial uncertainty and shape-shifting elements of the scene. On the other hand, the BEV grid offers many advantages to represent a complex driving scene ([section 2.3.3](#)). It is a compressed discretization of the 3D space, it can model a wide range of objects and their dynamic, and it naturally supports the task of planning. Nevertheless, online prediction of BEV semantic maps requires complex operations such as extracting and fusing information from multiple cameras, and projecting it into a common top-view grid. Inspired by recent advances in BEV estimation methods ([section 2.3.3](#)), we propose to learn the mapping from camera images to the BEV occupancy grid. However, existing methods for these operations rely on error-prone geometric operations or expensive direct dense mapping between image pixels and pixels in BEV. Instead, we propose to use an intermediate, compact, implicit representation to aggregate and fuse information within and across multiple cameras. The resulting internal representation of the scene is then reprojected in the BEV space to segment vehicles and driveable areas. We

also show that we can efficiently aggregate information over time into this implicit representation, paving the way for systems able to learn an internal representation of the world ([section 2.3.4](#)).

Chapter 3

Monocular metric depth estimation with a few-beam LiDAR

We have seen in [chapter 2](#) that *depth prediction* is often used to achieve a 3D spatial understanding of the surroundings, something necessary to plan and act safely [[Phillion and Fidler, 2020](#); [Srikanth et al., 2019](#); [Zeng et al., 2019](#)]. Depth information can be directly acquired using multiple LiDAR (32 beams or more), a sensor setup typical of expensive “over-engineered” Level-5 vehicles. Alternatively, camera-only methods offer a less costly approach by relying on a cheap sensor that is now ubiquitous on passenger vehicles with modern ADAS.

Nonetheless, self-supervised and trained with a view reconstruction objective, this kind of vision-based approach suffers from two major drawbacks. First, cameras on automated vehicles are often organized as a ring to provide a 360° view of the surroundings, leaving only very small overlaps between views, not enough to leverage stereographic principles. In such a monocular setup, there is an infinite number of 3D scenes that can explain the 2D projection of the image. This poses an *ambiguity of scale*, that is to say, the actual size, in meters, of an object cannot be determined from the image only. Secondly, the view reconstruction objective typically assumes a rigid scene, meaning that the scene is static. This assumption is often unrealistic in real-world scenarios and can result in the depth of moving objects dangerously overestimated. This problem is commonly referred to as the ‘infinite-depth’ problem and mostly happens for cars moving in front of the ego-vehicle, a situation typical of traffics, making existing methods unsafe to deploy ‘in-situ’.

In this chapter, we propose a new alternative for dense *metric* depth estimation by combining a monocular camera with a light-weight LiDAR, e.g., with 4 beams, typical of today’s automotive-grade mass-produced laser scanners. We introduce a novel framework, to estimate dense depth maps from monocular images with the help of “touches” of LiDAR. This method, called *LiDARTouch* has been submitted in 2020, and published in 2023, in the scientific journal CVIU [[Bartoccioni et al., 2023](#)]. We show that the use of a few-beam LiDAR alleviates scale ambiguity and infinite-depth issues that camera-only methods suffer from. We also demonstrate that methods from the fully-supervised depth-completion literature can be adapted to a self-supervised regime with a minimal LiDAR signal. At the time of submission, our LiDARTouch framework achieves new state-of-the-art in self-supervised depth estimation on the KITTI dataset, thus supporting our choices of integrating the very sparse LiDAR signal with other visual features.

To enable comparison with our work in the future, the code for our learning system and the data processing steps have been publicly released at <https://github.com/F-Barto/LiDARTouch>.

TABLE 3.1: **High-level positioning of LiDARTouch vs depth estimation and depth completion methods.** Our LiDARTouch framework addresses critical weaknesses of self-supervision depth estimation approaches, while being cheaper and far more scalable than fully-supervised depth completion methods.

Approach	Input	Supervision		Strengths (S) and Weaknesses (W)
		Depth regression	Photo. reconstr.	
Depth estimation	Image	No	Yes	S: scales well (self-supervised, very cheap sensor) W: relative depth, catastrophic estimations (moving objects)
Depth completion	Image and dense LiDAR	w.r.t. dense GT depth	No	S: metric depth, very good performance W: scales poorly (expensive sensors and GT annotations)
LiDARTouch (ours)	Image and few-beam LiDAR	w.r.t. few-beam LiDAR	Yes	S: scales well (self-supervised, cheap sensors) S: metric depth, good performance

3.1 Introduction

Accurately estimating depth in scenes is a prerequisite for a wide range of computer vision tasks, from computing semantic occupancy grids [Lee and Medioni, 2016; Ng et al., 2020] to object detection without labels [Deng et al., 2017; Koestler et al., 2020] and multi-modal unsupervised domain adaptation [Jaritz et al., 2020]. In particular, autonomous systems require a 3D understanding of their surroundings to plan and act safely, and the capacity to estimate depth is central to achieving this [Phillion and Fidler, 2020; Srikanth et al., 2019; Zeng et al., 2019]. As already stated in section 2.3.1, two main lines of approach exist to infer depth in a scene, depending on the available data: LiDAR-based completion and camera-only estimation methods. LiDAR-based depth *completion* methods rely on one or multiple *dense* LiDARs (e.g., 32 or 64 beams) [Jaritz et al., 2018; Park et al., 2020; Tang et al., 2020; Xu et al., 2019] to physically capture most of the geometry of the scene and interpolate the rest of the scene structure from the camera RGB signal. However, these approaches are so far unfit for automotive-grade settings, as they rely on expensive sensors — often costing more than a car alone — and require a rich supervisory signal for training, composed of 64-beam LiDAR point clouds densely accumulated over time. An alternative is explored by camera-only methods that predict dense depth maps with either stereo [Chang and Chen, 2018; Kendall et al., 2017] or monocular [Casser et al., 2019a; Godard et al., 2019, 2017; Guizilini et al., 2020a,b; Kuznietsov et al., 2017; Mahjourian et al., 2018; Wang et al., 2018; Yin and Shi, 2018; Zhou et al., 2017] setups. These models address the task of depth *estimation* and, contrary to the depth completion setup, do not leverage LiDAR point clouds. While such methods are appealing, as they rely on much cheaper and versatile sensors, monocular approaches suffer from ambiguity in the map scale they produce: most of them can only generate *relative* depth maps, i.e., up to an unknown global scaling factor, which makes them unusable in a real-world setting.

Moreover, their predictions can be catastrophic for objects with no relative motion with respect to the ego-camera, e.g., vehicles in front, which are likely estimated at infinite depth [Casser et al., 2019a; Godard et al., 2019; Guizilini et al., 2020a; Mahjourian et al., 2018; Wang et al., 2018; Yin and Shi, 2018; Zhou et al., 2017]. Lastly, they are critically impeded by low-light conditions (at night or indoors) and adverse weather (in heavy rain, dense fog, or snow storm) [Gruber et al., 2019].

In this thesis, we propose the LiDARTouch framework, where dense *metric* depth is estimated by combining a monocular camera with a *minimal* sparse LiDAR input (e.g., 4 beams). Our motivations to use a sparse LiDAR input are diverse. First, from a practical perspective, 4-beam laser scanners are currently embedded in consumer-grade vehicles and they are a hundred times less expensive than their dense (64-beam) counterparts. Second, we expect that such a LiDAR signal, although extremely sparse, can provide valuable cues for monocular depth estimation, thus alleviating scale-ambiguity and infinite-depth problems. Third, we hypothesize that a light LiDAR touch will result in the overall model correctly estimating the depth of moving objects, notably cars, alleviating the infinite-depth issue. Finally, from a security perspective, such an approach makes it difficult to attack the camera signal alone [Yamanaka et al., 2020], due to a form of data redundancy between the camera and LiDAR.

Leveraging recent advances in monocular depth estimation [Godard et al., 2019; Guizilini et al., 2020a; Watson et al., 2019; Zhou et al., 2017], our approach is *self-supervised*. This setting is significantly less data-hungry than the fully-supervised alternative, which requires densified and stereo-filtered depth maps as ground truth [Fu et al., 2018; Jaritz et al., 2018; Park et al., 2020; Tang et al., 2020; Xu et al., 2019]. We emphasize that this self-supervised learning setting, combined with the fact that it only involves widely available and low-priced sensors, makes the overall approach particularly scalable. Indeed, it becomes possible to estimate dense and metric depth maps on datasets and domains lacking depth ground truth [Caesar et al., 2020; Chang et al., 2019; Sun et al., 2020]. Moreover, from an industrial perspective, the LiDARTouch framework naturally scales with the data acquired by a vehicle fleet without the need for any annotation. Under this new regime, we propose the adaptation of recent methods from the two aforementioned streams of approaches for inferring depth. On the one hand, we adapt fully-supervised depth completion methods, namely ACMNet [Zhao et al., 2021] and NLSPN [Park et al., 2020], to a much sparser LiDAR using our self-supervised setup. On the other hand, we strengthen the very embodiment of self-supervised monocular camera-only methods, namely Monodepth2 [Godard et al., 2019], to integrate the new complementary LiDAR information. We then perform an extensive study on the contribution brought by the sparse LiDAR signal at different levels as: (1) an additional input, (2) a new information source to estimate better poses, and (3) a form of self-supervision. A high-level positioning of LiDARTouch with respect to depth estimation and completion approaches is summarized in [table 3.1](#).

To evaluate the adapted models and validate our hypotheses, we propose a novel training and evaluation protocol on the KITTI dataset [Geiger et al., 2012] which includes the degradation of the raw 64-beam LiDAR data to obtain 4 beams. We also propose a new metric to quantitatively measure the infinite-depth problem. This allows us to verify one of our core hypotheses that the use of very limited LiDAR information corrects infinite-depth degeneracies of camera-only methods. In comparison to depth completion methods, our LiDARTouch framework overcomes the need for depth ground truth and leads to highly improved results with respect to approaches that are naïvely adapted to the self-supervised setting. In addition, we show that it is possible to successfully adapt architectures from the depth completion literature, as well as camera-based depth estimation methods, into a unified framework that alleviates problems from which these two lines of approaches suffer.

We make the following contributions:

1. We propose LiDARTouch, a new *self-supervised* depth estimation framework, where a *minimal* LiDAR and a monocular camera are available without access to any ground-truth depth annotations. This configuration is close to *in situ* conditions of today’s vehicles, which is seldom addressed in other works.
2. We demonstrate that models trained within our LiDARTouch framework close the performance gap between self-supervised monocular depth estimation and fully-supervised

- depth completion learning schemes, proving that the need for ground-truth acquisition and costly sensors can be alleviated.
3. We show that models trained within our LiDARTouch framework do not suffer from critical scale-ambiguity and infinite-depth issues, in contrast to camera-only models. We evaluate this a novel metric to quantitatively measure the infinite-depth issue for the first time in the literature.
 4. We demonstrate that LiDARTouch is a versatile learning framework by successfully applying it to a range of network architectures: Networks from the depth-completion literature are revamped to work with very sparse LiDAR instead of dense ones and camera-only models are adapted to integrate LiDAR data.
 5. We study the influence of LiDAR inputs at each stage of our framework extensively. Our experiments show that integrating sparse LiDAR in a self-supervised scheme is not trivial. We provide key insights for the community on how the fusion scheme, the pose method and the supervisions interact.

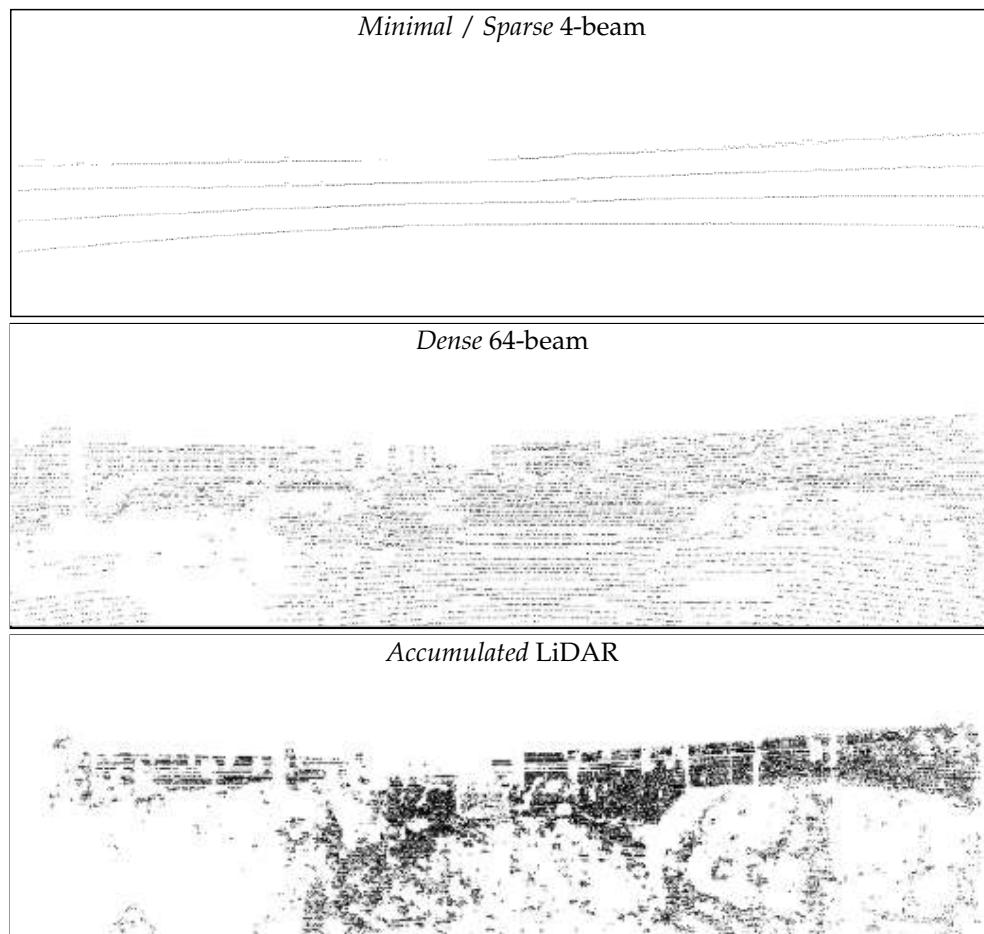


FIGURE 3.1: **Different LiDAR densities.** *Dense* 64-beam point clouds are typically used as the input of depth completion approaches, which are supervised with *accumulated* LiDAR seen as ground truth (GT). These point clouds are far denser than the *minimal* LiDAR we use. Note that LiDAR data is often not available in the upper part of the scenes.

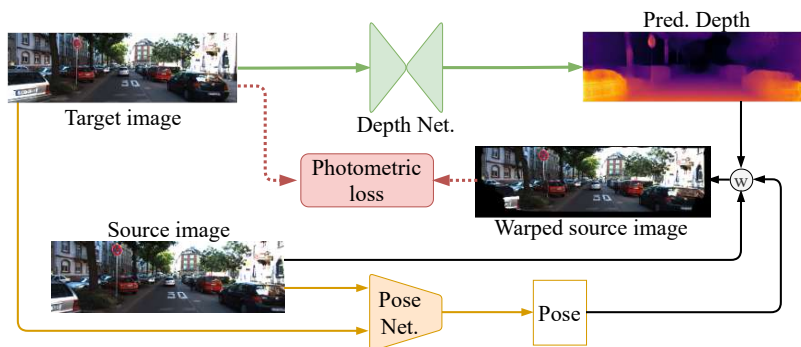


FIGURE 3.2: **Illustration of the self-supervised image-only depth estimation framework.** This figure shows the classical learning system from self-supervised image-only depth estimation literature, e.g., SfMLearner [Zhou et al., 2017] or Monodepth2 [Godard et al., 2019]. The model is trained to resynthesize the target image from (i) the neighboring source images with different viewpoints, (ii) the estimated depth of the target image, and (iii) the relative change of pose between the target and source views. The circled \otimes denotes the image warping operation from the estimated pose change and target depth map.

3.2 Related work

In the remainder of this thesis we refer to a LiDAR as *dense* if it has more than 32 beams, and call it *sparse* or *minimal* otherwise. Depth ground-truth, required by fully-supervised methods, is obtained from a dense LiDAR signal, accumulated over several sweeps. A camera stereo setup is then used to remove trail artifacts from moving objects. We will refer to such densified point-cloud data as *accumulated* LiDAR. These three density levels are illustrated in fig. 3.1. We now detail the two lines of approaches related to our work: camera-only monocular self-supervised methods and LiDAR-based fully-supervised depth completion systems.

Monocular self-supervised methods. In a fully- or semi-supervised setting, several models estimate depth in a camera-only monocular setup [Amiri et al., 2019; Fu et al., 2018; Kuznetsov et al., 2017], but acquiring depth ground truth for outdoor environments at scale is challenging and expensive. To overcome this issue, a few camera-based works [Casser et al., 2019a; Godard et al., 2017; Zhou et al., 2017] propose a *self-supervised* alternative to the use of ground-truth depth. Leveraging a set of consecutive frames, this paradigm predicts the depth for one of them and the relative changes in pose across nearby views. The model is trained by minimizing a photometric reconstruction error defined over these views (fig. 3.2). Two important issues with such approaches hinder their widespread usage: the scale ambiguity of the produced depth maps and the infinite-depth problem.

The *scale-ambiguity* problem stems from the view synthesis formulation being ill-posed. The formulation is scale ambiguous, as the target view can be correctly reconstructed regardless of the scale of the prediction. As a consequence, estimated depth maps are *relative* — up to an unknown global scaling factor — and models thus need additional supervision to accurately estimate a *metric* depth. Several self-supervised approaches rely on ground-truth LiDAR signal to scale their depth estimation at test time [Casser et al., 2019a; Godard et al., 2019; Mahjourian et al., 2018; Wang et al., 2018; Yin and Shi, 2018; Zhou et al., 2017]. Alternatively, the recent PackNet model [Guizilini et al., 2020a] proposes to automatically scale estimations with additional constraints imposed by the instantaneous velocity of the ego-vehicle. Some works have also moved to a stereo setup to disambiguate the scale factor, using additional information, at train time only [Godard et al., 2017; Groenendijk et al.,

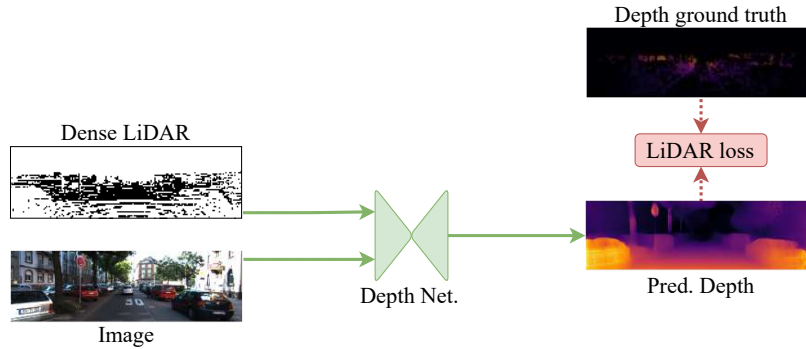


FIGURE 3.3: **Illustration of the fully-supervised depth completion framework.** This figure summarizes the depth completion pipeline, e.g., models ACMNet [Zhao et al., 2021] or NLSPN [Park et al., 2020], which employs a multi-modal depth prediction network that is learned by regressing a provided ground-truth depth.

2020] or also at run time [Chang and Chen, 2018; Cheng et al., 2019; Kendall et al., 2017], thus abandoning the monocular setup.

The second issue of infinite depth arises when objects move at the same speed as the camera. In this common situation, a trivial solution for the model is to predict that these objects are infinitely far and big, as they do not change in appearance through time [Gordard et al., 2019; Guizilini et al., 2020a; Zhou et al., 2017]. Recent proposals to address this problem exploit semantic segmentation of classes known to be often dynamic (e.g., cars, trucks) [Casser et al., 2019a,b], or automatically prune the dataset by removing these objects [Guizilini et al., 2020b]. The robustness of both these approaches to novel test scenarios, however, remains unclear.

In our work, we build on camera-only methods to additionally integrate LiDAR information and show that: (i) very few direct depth measures suffice to have a metrically-scaled dense depth estimation, and (ii) the infinite-depth issue can be partially or completely solved with the use of LiDAR input, depending on its resolution and position, without any additional assumptions.

Depth completion methods typically estimate a dense depth map from raw LiDAR measurements. Current deep-learning based methods for depth completion [Jaritz et al., 2018; Kumar et al., 2018; Ma and Karaman, 2018; Park et al., 2020; Tang et al., 2020; Xu et al., 2019; Zhao et al., 2021] usually learn to regress ground-truth depth maps in a fully-supervised setup (fig. 3.3). Such approaches generally operate over RGB and LiDAR inputs.

A popular approach is to use one encoder per modality and fuse them at each resolution scale [Guizilini et al., 2021; Tang et al., 2020] or at the feature bottleneck only [Jaritz et al., 2018]. An other option is early fusion, where both modalities are concatenated at the very beginning of the architecture [Ma et al., 2019; Park et al., 2020; Xu et al., 2019]. Some fusion modules, as the one of GuideNet [Tang et al., 2020], only considers the image as a guiding signal for the LiDAR features. This assumes that the LiDAR input is sufficient, i.e., high-resolution, for estimating depth, and thus unsuitable for our case. This limits the approach [Tang et al., 2020] to estimate depth from high-resolution 64-beam LiDAR both at train and run time, making it incomparable to ours as we do not have access to such data. On the contrary, the SAN architecture [Guizilini et al., 2021], can handle various levels of LiDAR sparsity with sparse convolutions. Alternatively, networks like ACMNet [Zhao et al., 2021] and NLSPN [Park et al., 2020] propagate sparse LiDAR features into image features where depth measurements are not available. ACMNet [Zhao et al., 2021] uses a multi-scale co-attention-guided graph propagation strategy for depth completion. It propagates

the sparse and irregularly distributed LiDAR measurements through a nearest-neighbor encoding. In addition, it uses a symmetric gated fusion strategy to fuse multi-modal contextual information throughout the decoder. The NLPSN architecture [Park et al., 2020] jointly estimates an initial depth map, a pixel-wise confidence and non-local affinity kernels. This initial depth map is iteratively refined with the input LiDAR features using the predicted confidence map and affinity kernels.

All the aforementioned depth completion methods employ a 64-beam input LiDAR and are trained with accumulated LiDAR as supervision. Here, most of the scene structure is available and the task amounts to color-guided depth interpolation. This design prevents these works from being easily adapted to new domains. Indeed, the acquisition of ground-truth data is expensive and not scalable, as it is obtained from high-resolution LiDARs and stereo cameras. In contrast, our work specifically focuses on minimal 4-beam LiDAR directly, with no densely accumulated LiDAR data as supervision. We emphasize that in this very sparse 4-beam regime, almost no structural information can be directly extracted for the input signal. The task we propose is then more akin to depth estimation than depth completion.

A closely related work to ours is the model of [Ma et al., 2019], which also uses LiDAR as a *supervisory* signal in a monocular self-supervised setting. LiDAR and camera signals are merged through an early fusion and the change of pose is estimated by solving a Perspective- n -Point problem. However, their setup is different from ours. Their study focuses on the dense depth completion regime, i.e., with a 64-beam LiDAR, while we work on depth estimation with a minimal 4-beam LiDAR. Moreover, they do not compare against other existing architectures in the self-supervised setting. In contrast, we perform thorough evaluations with existing work by adapting camera-only and depth completion methods to our extremely scarce LiDAR regime. Additionally, we propose a different supervision scheme and the use of multiple views in photometric reconstruction. These choices lead to a substantial improvement on the KITTI dataset. Finally, we provide in-depth analyses of the impact brought by the LiDAR signal at different levels.

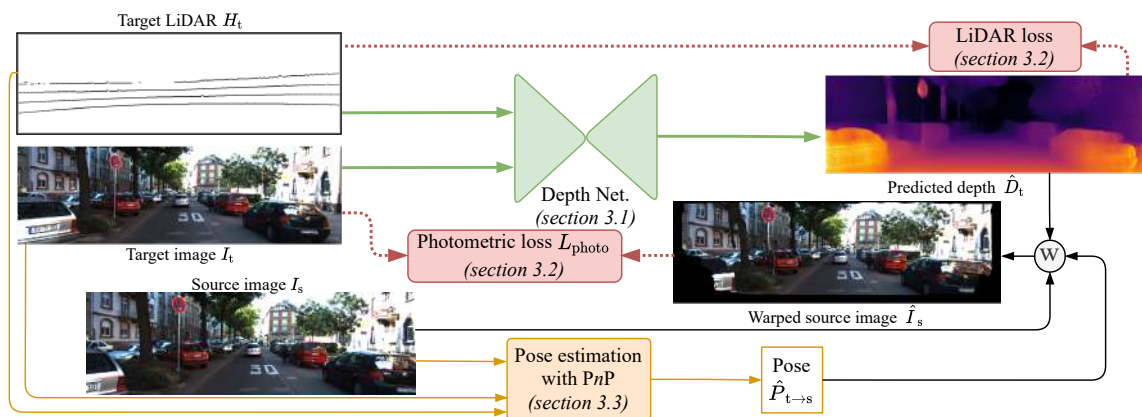


FIGURE 3.4: **Overview of our LiDARTouch learning framework.** The proposed framework leverages ideas from both the camera-only depth estimation approach (illustrated in Figure 3.2) and fully-supervised depth completion methods (illustrated in Figure 3.3). In LiDARTouch, the light touch of LiDAR is integrated at three different stages: as an input of the depth network, as a self-supervision signal, and to estimate a scaled pose.

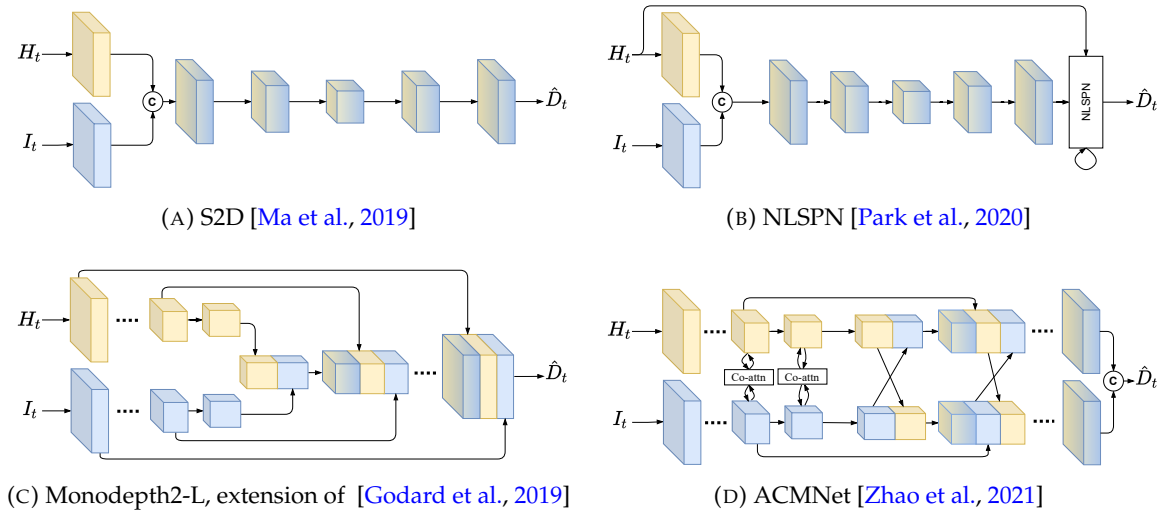


FIGURE 3.5: **Depth networks with different image-LiDAR fusion strategies.** We depict early (A), hybrid (early and late for B) as well as multiscale (C and D) fusion-based architectures. Volumes in yellow indicate LiDAR feature tensors and blue ones are image feature tensors. We indicate the mixing of modalities with a color grading of the two colors on the volumes. The architecture (C) is our extension of [Godard et al., 2019] to make it operate over *minimal* LiDAR input. We denote the concatenation operator by \oplus .

3.3 LiDARTouch framework

This section is organized into three parts, each corresponding to a different and complementary use of the light LiDAR signal. In section 3.3.1, we present the architecture of the depth network, shown in green in fig. 3.4, which estimates depth by fusing the monocular image with the sparse LiDAR point cloud. In section 3.3.2, we detail the self-supervision objectives involving a photometric reconstruction along with a LiDAR self-supervision, as illustrated in red in fig. 3.4. Lastly, section 3.3.3 introduces methods to estimate the relative change of pose between the source and target views, depicted by the orange part of fig. 3.4.

3.3.1 Depth network

The core of our depth estimation system is a neural network taking the target image I_t coupled with H_t , the LiDAR data projected in the image plane, as input, and predicting a depth map \hat{D}_t . Given the multi-modal nature of the input, our depth network employs a fusion strategy, that can be either early or multi-scale. In this thesis, we consider four different architectures that are illustrated in fig. 3.5. Three of them are from the recent depth-completion literature, namely NLSPN [Park et al., 2020], S2D [Ma et al., 2019] and ACMNet [Zhao et al., 2021]. The fourth one, we refer to as Monodepth2-L, is an extension of the camera-only model Monodepth2 [Godard et al., 2019] to operate over the additional LiDAR input (we provide details of this extension in section 3.8).

The two architectures NLSPN [Park et al., 2020] and S2D [Ma et al., 2019], illustrated in Figures 3.5b and 3.5a respectively, employ an early-fusion strategy, combining image and LiDAR features from the start, through concatenation. Early fusion directly mixes features from both modalities, thus potentially enabling richer interactions across them. The NLSPN architecture additionally re-injects the LiDAR signal at the end of the processing, as a late refinement strategy to mitigate signal degradation due to normalization layers.

In contrast, Monodepth2-L and ACMNet architectures, represented in Figure 3.5c and 3.5d respectively, use a multi-scale fusion. They both encode LiDAR and visual data separately so that these modalities are processed differently and their learned features are progressively integrated together. This design merges modalities more carefully than the early-fusion strategy, which is desirable as visual and LiDAR inputs carry complementary semantics. The two encoders, based on ResNet-18 [He et al., 2016], are independent and modality-specific features are fused with a series of concatenations. ACMNet, on the other hand, employs a more sophisticated co-attention strategy to mutually guide the features in the encoders and mix the features in the decoders to finally fuse them into one prediction.

3.3.2 Self-supervision objectives

Our challenging setting, where depth ground truth is unavailable for training the model, prevents the depth network architecture to be supervised directly. We address this by training the network under the supervision of two combined objectives. The first one, photometric reconstruction L_{photo} , is inspired by recent advances in self-supervised camera-only monocular depth estimation [Godard et al., 2019, 2017; Zhou et al., 2017]. However, as discussed in section 3.2, training with this objective alone leads to scale and infinite-depth issues. Consequently, we leverage a LiDAR self-reconstruction objective, which uses sparse yet complementary LiDAR information to mitigate these issues.

Self-supervised photometric reconstruction L_{photo} . We recall that the photometric reconstruction problem is a surrogate task aimed at resynthesizing a target image, given neighboring source images with different viewpoints [Godard et al., 2019; Ma et al., 2019; Zhou et al., 2017]. Solutions to this task build on optimization approaches for disparity, motion and depth estimation without learning, based on photo-consistency. The central idea is to combine pose and depth predictions to project a neighboring source image into the target view. The underlying intuition is that to accurately resynthesize the target view from the source one, both the depth and pose estimation must be accurate.

Formally, the *target* image I_t is considered with a set S of *source* images I_s in its temporal vicinity. First, the depth network predicts the dense depth map \hat{D}_t for the target image I_t . Second, the relative changes of pose $\hat{P}_{t \rightarrow s}$ between the target and source views are estimated — we detail this in section 3.3.1. One pose transformation $\hat{P}_{t \rightarrow s} = \begin{pmatrix} \hat{R} & \hat{r} \\ 0 & 1 \end{pmatrix} \in \text{SE}(3)$ is estimated for each source image $I_s \in S$, where \hat{R} is a rotation matrix and \hat{r} the translation component. Given the estimates of depth and pose, and the camera intrinsics K , a source image I_s can be warped via a differentiable geometric transformation into synthetic image \hat{I}_s in the target view. More precisely, for homogeneous coordinates p_t of a pixel in the target image, the projected coordinates p_s in the source image are computed with:

$$p_s \simeq K \hat{P}_{t \rightarrow s} \hat{D}_t(p_t) K^{-1} p_t. \quad (3.1)$$

For a pair (I_s, I_t) of source-target images, the reconstructed image \hat{I}_s is enforced to match the target image I_t by a pixel-wise image reconstruction error based on both an L_1 intensity loss and a structural similarity (SSIM) loss [Loza et al., 2006]. Note that this formulation assumes Lambertian surfaces.

More formally, at a given pixel location p , this loss reads:

$$L_{\text{photo}}(p) = \min_{I_s \in S} \left\{ \frac{\alpha}{2} (1 - \text{SSIM}(I_t, \hat{I}_s)(p)) + (1 - \alpha) |I_t(p) - \hat{I}_s(p)| \right\}, \quad (3.2)$$

where α is a hyper-parameter balancing the contributions of the two terms. Moreover, taking the minimum value over all source images $I_s \in S$ limits the impact of errors resulting

from occlusions and disocclusions in the scene due to motion of the ego-car and/or of the other scene elements [Godard et al., 2019]. To take into account objects with no motion with respect to the ego-car, this loss is only applied to pixels whose appearance between frames varies [Godard et al., 2019].

LiDAR self-supervision. As detailed in section 3.2, a model solely trained with the photometric reconstruction loss L_{photo} suffers from a scale-ambiguity issue and may be affected by the infinite-depth problem. In the following, we describe the new role of the low-density input LiDAR as a supervisory signal to mitigate this problem. We assume that this complementary information source can provide minimal-yet-crucial cues to disambiguate the estimated depth, at a global scale level and especially for moving objects. Furthermore, a sparse depth signal can refine the photometric supervision for small objects, thus improving overall performances [Watson et al., 2019]. Inspired by the depth completion and the stereo depth estimation literature, we consider three different ways of using LiDAR as a supervisory signal: a straightforward L_1 regression along with two refinements that either control the interference with the photometric reconstruction or take into account the inherent noise of the LiDAR signal.

First, we consider a naïve self-supervision scheme, an L_1 loss for all pixels having a LiDAR measurement, in addition to the photometric loss L_{photo} :

$$L_{\text{naïve}}(p) = \begin{cases} |\hat{D}_t(p) - H_t(p)| + L_{\text{photo}}(p) & \text{if } H_t(p) > 0, \\ L_{\text{photo}}(p) & \text{otherwise,} \end{cases} \quad (3.3)$$

where p is an index over the pixels, \hat{D}_t the estimated depth and H_t the input LiDAR projected in the target image plane. The latter being sparse, not all pixels have LiDAR data available; we use the encoding $H_t(p) = 0$ for such pixels.

Second, we consider the *masked* self-supervised objective proposed in [Ma et al., 2019]. It makes the LiDAR regression and the photometric loss exclusive by masking-out the photometric loss L_{photo} on pixels with a LiDAR measurement. Denoting L_{masked} as this loss, it is given by:

$$L_{\text{masked}}(p) = \begin{cases} |\hat{D}_t(p) - H_t(p)| & \text{if } H_t(p) > 0, \\ L_{\text{photo}}(p) & \text{otherwise.} \end{cases} \quad (3.4)$$

This loss is similar to $L_{\text{naïve}}$ but avoids potential conflicts between the photometric and LiDAR reconstructions.

Lastly, inspired by [Watson et al., 2019], we also introduce the *hinted* self supervision, L_{hinted} , that takes into account the inherent noise of the LiDAR signal. Despite being a direct depth measurement, raw LiDAR signal is noisy for a number of reasons, including potentially imprecise calibration, approximated projection, and the fact that the camera and LiDAR are not exactly positioned at the same place, which results in objects observable by one but hidden to the other. Therefore, the loss L_{hinted} integrates the LiDAR self-supervision only where image reconstruction is more precise by using the LiDAR signal instead of the estimated depth. More precisely, two versions of the photometric contribution of the pixel are computed: the regular pixel-wise photometric loss L_{photo} , using the estimated depth map \hat{D}_t in eq. (3.1), and L_{photo}^H using the input projected LiDAR H_t instead of \hat{D}_t in eq. (3.1). Then we only supervise with the LiDAR reconstruction when $L_{\text{photo}}^H < L_{\text{photo}}$. The objective is thus:

$$L_{\text{hinted}}(p) = \begin{cases} |\hat{D}_t(p) - H_t(p)| + L_{\text{photo}}(p) & \text{if } L_{\text{photo}}^H(p) < L_{\text{photo}}(p) \\ L_{\text{photo}}(p) & \text{otherwise.} \end{cases} \quad (3.5)$$

3.3.3 Pose estimation

The formulation of the photometric reconstruction involves the change of pose $\hat{P}_{t \rightarrow s}$ between the target image I_t and source view I_s for the source image warping. A first possibility, which is widely used in monocular self-supervised depth estimation [Casser et al., 2019a; Godard et al., 2019; Guizilini et al., 2020a; Zhou et al., 2017], uses a so-called *pose network* jointly trained with the depth network. However, due to the monocular ambiguity, this approach can only estimate a relative pose and thus relative depth maps, which then must be rescaled by an unknown factor. Instead, we explore another way to estimate a metric pose, by leveraging the LiDAR information and solving a Perspective- n -Point problem [Gao et al., 2003; Lepetit et al., 2009]. As such, depth estimation should also align to a real-world scaling.

Perspective- n -Point (PnP). The PnP problem originally seeks the absolute pose of a camera given a set of 3D points and their corresponding 2D image projections. In our case, we use the PnP formulation to estimate the change of pose between the target and source views, i.e., given the target image I_t and LiDAR measurements, as well as the source image I_s .

First, pairs of pixels (p_t, p_s) matching in both views I_t and I_s are found using the SIFT descriptor [Lowe, 2004] based on a DoG keypoint detector. Then, the sole pairs for which p_t has a LiDAR measurement are considered. This gives us the pairs of 3D-2D points, where points p_t are complemented with depth measurements and match the 2D points p_s of the source image I_s . Given these pairs, we can precisely estimate the *metric*-scaled 6D rigid transformation between the target and source poses by minimizing the cumulative projection error.

In challenging real-life situations, and especially when dealing with a 4-beam LiDAR, finding matching pixels that have LiDAR measurements can be arduous, making this method prone to errors. Hence, we follow [Ma et al., 2019] to remove outliers in the set of point correspondences by using RANSAC in conjunction with the PnP solving algorithm. When this filtering step is insufficient for the algorithm solving the PnP problem to converge, we discard the training sample.

3.4 Experimental protocol

The first component of our protocol is the dataset used for the experiments, namely KITTI [Geiger et al., 2012], and our preprocessing to reduce the raw 64-beam LiDAR to a 4-beam one (section 3.4.1). We then introduce baselines in section 3.4.2. Additional details are given in appendix A.

3.4.1 Dataset and evaluation metrics

To train models in our LiDARTouch framework, we need a dataset that provides a camera stream with aligned sparse LiDAR data for training. We also require this dataset to have ground-truth depth data with an associated benchmark to assess and compare our test performances. We are aware of only one dataset matching both of these requirements, namely KITTI. It contains 1.5 hours of recorded driving sessions in urban environment from a video stream synchronized with LiDAR data. Depth ground truth is available: it is derived from dense LiDAR signals accumulated over five sweeps and stereo filtered. Overall, we use this dataset to train and evaluate the quality of the predictions of our framework, and to compare against baselines and variants. On the KITTI dataset [Geiger et al., 2012], we use the so-called Eigen split [Eigen et al., 2014] for train, val and test with a minor modification for the val and test. The ground-truth LiDAR of [Uhrig et al., 2017] is

not available for some of the frames of the Eigen splits (fewer than 10). Following common practice [Godard et al., 2019; Guizilini et al., 2020a], we removed them from the val and test splits. Thus, the total number of examples are 22537, 873 and 652 respectively for the train, val and test sets.

The LiDAR data provided in KITTI is obtained with high-end 64-beam sensors, appropriate for *evaluating* our self-supervised models, but much denser than what is expected to *train* our LiDARTouch framework. Consequently, we perform a filtering step to extract 4 beams out of the raw 64-beam LiDAR data. To conform with prior works [Guizilini et al., 2019; Jaritz et al., 2018; Ma et al., 2019] and better compare with them, we sample LiDAR beams uniformly: 1 beam is kept every 16. Note that with such a sampling, while 4 beams are extracted, only three beams effectively project onto the image plane as one beam falls out of the considered visual region.

Evaluation metrics. Evaluation is conducted against accumulated ground-truth LiDAR obtained following [Uhrig et al., 2017], with the metrics defined in [Eigen et al., 2014]. This includes the absolute (Abs Rel) and square (Sq Rel) relative errors, the root mean square error (RMSE), and its log version (RMSE_{\log}), as well as precision-under-threshold metrics measuring the percentage of depth predictions \hat{D} close enough to the ground-truth depth D , in the sense of the value $\delta := \max(\frac{\hat{D}}{D}, \frac{D}{\hat{D}})$ being under a user-defined threshold. Following [Eigen et al., 2014], we consider three thresholds: $\delta < 1.25$, $\delta < 1.25^2$ and $\delta < 1.25^3$.

3.4.2 Notations, ablations and external baselines

Notations. To refer to the network architecture, independently of the rest of the learning framework, we use Monodepth2, Monodepth2-L, NLSPN, ACMNet and S2D. When we refer to whole models, i.e., architectures trained under the LiDARTouch framework, we append the ‘LiDARTouch’ prefix. For example, we note ‘LiDARTouch-ACMNet’ when we adapt the ACMNet architecture into the LiDARTouch framework.

For clarity, the inputs and the supervision schemes that are employed by the models are recalled in the tables of the experiments section. The input of each depth prediction model includes an image (noted ‘ \mathcal{I} ’) and, optionally, a sparse 4-beam LiDAR point cloud (‘ \mathcal{L}^4 ’). We considered the following supervisions strategies: self-supervised photometric reconstruction (‘ \mathcal{P} ’) associated to loss eq. (3.2), supervised LiDAR ground-truth regression with L_1 loss (‘ \mathcal{L}_{gt} ’), or LiDAR self-supervision (‘ \mathcal{L}_4 ’) with one of the three options in Eqs. (3.3), (3.4), or (3.5).

Ablation: Pose estimation with a pose network. In section 3.3.3, we presented the PnP algorithm, which estimates metric pose changes from source to target views. To highlight the gains enabled by the use of the extra LiDAR information for computing the pose, we experiment by training a *pose network* instead, a widely used component of monocular depth estimation models [Casser et al., 2019a; Godard et al., 2019; Guizilini et al., 2020a; Zhou et al., 2017]. For each target-source image pair, the pose network outputs the 6D rigid transformation between views. It is differentiable and trained jointly with the depth network. When only trained with the photometric error (eq. (3.2)), the 6D transformation is estimated up to a scale factor due to the monocular ambiguity. This results in a relative depth estimation requiring to be rescaled by the LiDAR depth ground-truth median value (not available in our case).

A solution is to use data from the IMU/GNSS to supervise the pose estimation scale. In the context of depth estimation, such an approach has been explored by [Guizilini et al., 2020a]. Formally, we first obtain the approximate change in pose between the source and target views ($P_{t \rightarrow s}$) from integrated inertial measurements. Then, we extract its translation

component r and make the predicted pose translation component \hat{r} regress its magnitude:

$$L_{\text{imu}} = \left| \|r\|_2 - \|\hat{r}\|_2 \right|. \quad (3.6)$$

As for a given pose there is a unique depth minimizing eq. (3.2), constraining the pose’s magnitude to a metric scale forces the depth estimation to be metric as well.

Baselines: Monocular methods. We compare against state-of-the-art monocular self-supervised approaches such as SfMLearner [Zhou et al., 2017], Vid2Depth [Mahjourian et al., 2018], GeoNet [Yin and Shi, 2018], DDVO [Wang et al., 2018], Monodepth2 [Godard et al., 2019], PackNet-SfM [Guizilini et al., 2020a] and MonoViT [Zhao et al., 2022]. Note that these methods can only produce relative depth maps, as they use an unsupervised pose network, so they have to be rescaled using the ground-truth LiDAR. Comparisons with these methods is thus unfair, in their favor.

Additionally, we compare with methods that directly produce metric depth by leveraging additional supervision. This includes (1) DORN [Fu et al., 2018], a camera-only method fully-supervised by a dense LiDAR signal, (2) [Kuznietsov et al., 2017], a semi-supervised method using stereo reconstruction and dense LiDAR supervision, and (3) PackNet-SfM [Guizilini et al., 2020a] model supervised with IMU prior.

Baselines: Depth completion methods. We also compare against supervised depth completion methods, namely ACMNet [Zhao et al., 2021], NLSPN [Park et al., 2020] and S2D [Ma et al., 2019]. However, their original versions are not trained and evaluated on the same splits as monocular methods. We re-train and evaluate them on the Eigen split, in their fully-supervised setting but with only a 4-beam LiDAR input. Additionally, we also train and evaluate these depth completion methods when the depth ground truth is simply replaced by the 4-beam LiDAR input for supervision signal. We refer to this setting as ‘Naïve self-sup.’.

3.5 Influence of a touch of LiDAR

In this section, we validate setups where the depth network converges to a metric scale. In particular, in section 3.5.1, we disentangle the contributions brought by LiDAR with an ablation study on the three levels of integration presented in section 3.3: as a self-supervision signal, as a depth network’s input, and as additional information for pose estimation. We also investigate various combinations of LiDAR self-supervision schemes and depth networks in section 3.5.2.

3.5.1 Ablation of LiDAR

We begin with an ablation study to assess the contribution brought by sparse LiDAR at three different levels: supervision, input and pose. We define our LiDARTouch framework as using a PnP for pose estimation, LiDAR self-supervision (L_4) with the *masked* loss variant, and a bi-modal depth network (i.e., taking RGB and LiDAR as input). Models that belong to this framework are highlighted as light blue cells in table 3.2. For the sake of clarity, in this section we focus on the leftmost three columns for direct comparison with LiDARTouch. Other learning setups are discussed in detail in appendix A.2.

LiDAR as an input. First, we study the contribution brought by LiDAR when it is used as an input to the depth network in addition to the image signal. Results in the first column of table 3.2 show that the Monodepth2 architecture, which does not use LiDAR as input, is consistently outperformed by all the other bi-modal architectures leveraging LiDAR input.

TABLE 3.2: **Pose estimation ablation.** We report precision (%) under threshold ($\delta < 1.25$) on the KITTI test split; higher is better. As we are interested in *metric* depth estimations, contrary to common practice [Godard et al., 2019; Guizilini et al., 2020a], estimations are **not** rescaled with LiDAR GT. Light blue cells indicate configurations corresponding to our LiDARTouch framework. Some models are more difficult to train and indicated in light grey cells. In particular, ‘*’ implies that a rescaling of the pose was used for a stable training, and ‘+’ indicates that the LiDAR signal had to be dilated to avoid overfitting to the LiDAR input (more details in A.3). When using a pose network with photometric supervision only (dark gray cells), the estimation can only be *relative* and the scores are all below 1%. More details are provided in subsection 3.5.1.

Depth network		w/ PnP		w/ pose network			
		P+L ₄	P	P+L ₄	P	P+imu	P+L ₄ +imu
Monodepth2	LiDARTouch	86.1	86.2*	83.9	-	86.2	86.5
Monodepth2-L		96.9	96.2*	94.9	-	96.4	96.5
ACMNet		97.4	97.5	91.2 ⁺	-	27.0	95.3
NLSPN		95.9	96.8	94.2 ⁺	-	38.5	94.1 ⁺
S2D		96.2	96.4	93.9 ⁺	-	28.7	94.0 ⁺

These architectures achieve a relative improvement of 11-13% compared to Monodepth2. This validates the positive influence of integrating few-beam LiDAR as an input.

Self-supervision with the sparse LiDAR. Next, we study the impact of using a 4-beam LiDAR as a self-supervisory signal by removing it from the LiDARTouch framework, which leaves only the photometric loss (‘P’). This corresponds to the second column in table 3.2. Overall, the results support our claim that the use of LiDAR self-supervision improves or is similar in performance with respect to the photometric-only supervision schemes.

Although ACMNet, NLSPN and S2D architectures show slightly better performance when trained with PnP and the photometric-only loss, i.e., without any LiDAR self-supervision, they are severely affected by the infinite-depth issue (see section 3.7).

Moreover, when using the photometric loss alone (‘P’ in the table), Monodepth2 and Monodepth2-L are hard to train. Indeed, while PnP pose is metric by construction, the depth network is initialized randomly and has to converge to a metric scale with the photometric reconstruction as the sole learning signal. Without any precaution, we observe large numerical differences in scale at initialization between the pose and depth, which provoke unstable training for the depth network. To address this instability, we divide the translation component of the PnP pose by a factor α during training and multiply the depth prediction consequently at inference (details in appendix A.5). This procedure is inspired by the baseline (distance between the two cameras) scaling introduced in Monodepth2 for the stereo setting [Godard et al., 2019]. We indicate models that need to be trained using this strategy with ‘*’ in table 3.2. On the other hand, under the LiDARTouch framework, all depth networks train well without requiring training tricks.

Pose estimation with a sparse LiDAR. We now show that a precise computation of the change of pose is critical to estimate depth maps that are correctly scaled, and that a touch of LiDAR is beneficial for this purpose. To demonstrate this, we experiment by replacing PnP in our LiDARTouch setup with a pose network that does not use any LiDAR information, as detailed in section 3.4.2. This ablation of LiDARTouch corresponds to the third column, ‘P+L₄’ under ‘w/ pose network’, in table 3.2.

The main difference between these two setups is that PnP methods produce *metric* poses by construction, which left only metric depths as solutions to minimize the photometric loss. In opposition, the use of a pose network requires a joint alignment and convergence

TABLE 3.3: **Variants comparison of the LiDAR self-supervision.** RMSE metric (lower is better) on the Eigen test split of KITTI. Models are trained with photometric self-supervision (P) in conjunction with one of the three considered variants of minimal-LiDAR self-supervision (L_4). All models are trained with PnP for pose estimation.

Self-supervision	Monodepth2	Monodepth2-L	ACMNet	NLSPN	S2D
P + L_4 (naïve)	4.504	2.796	2.490	3.084	2.839
P + L_4 (hinted)	4.794	2.813	2.563	3.271	2.982
P + L_4 (masked)	4.517	2.696	2.504	3.014	2.776

to a metric scale between the depth and pose networks as they are both randomly initialized. While Monodepth2-L achieves this, it can be observed that the use of a pose network instead of PnP degrades performance up to 6% when compared to LiDARTouch. Above all, we observe a tendency for ACMNet, NLSPN and S2D to overfit the LiDAR signal (see [fig. A.1b](#) for an example).

We find that the multi-scale prediction and supervision during training of Monodepth2 and Monodepth2-L are key for the models not to overfit the sparse 4-beam LiDAR data. Indeed, supervision at the lowest scale (1:8) increases the number of pixels getting supervision from LiDAR as pixels with associated LiDAR signal are expanded due to the difference in scale.

Building on this observation, we propose a procedure to simulate this behavior in order to avoid LiDAR overfitting for mono-scale networks without changing their architectures. To simulate a LiDAR self-supervision at a lower scale, we apply a *dilation* morphological operation on the 4-beam LiDAR at the supervision level. This artificially increases the number of pixels receiving LiDAR supervision, albeit in a noisy manner, and enables the mono-scale depth networks ACMNet, NLSPN and S2D to produce globally coherent metric depth estimations. We report results of models trained with this procedure (indicated by ‘+’) in [table 3.2](#) and provide technical details as well as qualitative examples in [appendix A.3](#).

On the other hand, training under our LiDARTouch framework eliminates the need for such tricks. Indeed, results demonstrate that our LiDARTouch framework, using LiDAR as self-supervision, in input and in pose computation yields competitive performances for all the five architectures, a more stable training compared to any other configuration, and alleviates the infinite-depth problem as we will show in [section 3.7](#).

3.5.2 LiDAR self-supervision variants

We compare in [table 3.3](#) the variants for the LiDAR loss defined in [section 3.3.2](#), namely the *naïve* compound loss [eq. \(3.3\)](#), the *masked* one [eq. \(3.4\)](#), which prevents interferences with the photometric error, and the *hinted* loss [eq. \(3.5\)](#), which handles the noise of the LiDAR signal. These experiments are conducted for the four different depth networks considered in [section 3.3.1](#). Overall, averaged over all architectures, the *masked* version of the LiDAR loss achieves the best results, demonstrating the need to reduce interferences between the LiDAR and photometric supervisions. On the other hand, we observe that the *hinted* loss yields the worst results. We expected the *naïve* loss to have the worst performance as it does not consider the noise in LiDAR, but it appears that the control the *hinted* loss imposes is too strong and discards too many of the already scarce LiDAR measurements. Hence, it confirms that the *masked* LiDAR self-supervision is the most effective.

TABLE 3.4: **Comparison against monocular depth estimation methods.** Results are reported on the KITTI Eigen split [Eigen et al., 2014] with improved ground truth [Uhrig et al., 2017]. A few self-supervised methods produce relative-depth maps and their prediction must be rescaled using ground-truth information; this is identified by ‘*gt rescaled*’ in the table. Some of the methods also benefit from an extra pre-training, on ImageNet [Deng et al., 2009] or Cityscapes [Cordts et al., 2016], denoted with \circ or \star superscripts, respectively. The model *Monodepth2* in italic indicates our re-implementation of [Godard et al., 2019] without pre-training and post-processing. Input includes the image only (\mathcal{J}), or combined with the few-beam LiDAR point cloud (\mathcal{L}^4). Supervision includes photometric loss (\mathcal{P}), IMU prior (imu), stereo reconstruction (ste) and LiDAR supervision with either dense ground truth (\mathcal{L}_{gt}) or sparse 4-beam LiDAR (\mathcal{L}_4). A hyphen indicate that the score is not communicated by the authors of the method.

	Method	Input	Superv.	Abs Rel \downarrow	Sq Rel \downarrow	RMSE \downarrow	RMSE $_{\log}\downarrow$	$\delta < 1.25^\uparrow$	$\delta < 1.25^2^\uparrow$	$\delta < 1.25^3^\uparrow$
Sup.	DORN $^\circ$ [Fu et al., 2018]	\mathcal{J}	\mathcal{L}_{gt}	0.072	0.307	2.727	0.120	0.932	0.984	0.995
	[Kuznetsov et al., 2017] $^\circ$	\mathcal{J}	$\mathcal{L}_{\text{gt}}+\text{ste}$	0.089	0.478	3.610	0.138	0.906	0.980	0.995
Self-supervised	SfMLearner * [Zhou et al., 2017]	\mathcal{J}	\mathcal{P}	0.176	1.532	6.129	0.244	0.758	0.921	0.971
	Vid2Depth * [Mahjourian et al., 2018]	\mathcal{J}	\mathcal{P}	0.134	0.983	5.501	0.203	0.827	0.944	0.981
	GeoNet * [Yin and Shi, 2018]	\mathcal{J}	\mathcal{P}	0.132	0.994	5.240	0.193	0.883	0.953	0.985
	DDVO [Wang et al., 2018]	\mathcal{J}	\mathcal{P}	0.126	0.866	4.932	0.185	0.851	0.958	0.986
	<i>Monodepth2</i> (our reimplem.)	\mathcal{J}	\mathcal{P}	0.099	0.591	4.030	0.149	0.897	0.976	0.993
	<i>Monodepth2</i> $^\circ$ [Godard et al., 2019]	\mathcal{J}	\mathcal{P}	0.090	0.545	3.942	0.137	0.914	0.983	0.995
	PackNet-SfM * [Guizilini et al., 2020a]	\mathcal{J}	\mathcal{P}	0.071	0.359	3.153	0.109	0.944	0.990	0.997
	MonoViT * [Zhao et al., 2022]	\mathcal{J}	\mathcal{P}	0.067	0.328	3.108	0.104	0.950	0.992	0.998
	<i>Monodepth2</i> w/ IMU supervision	\mathcal{J}	$\mathcal{P}+\text{imu}$	0.110	0.729	4.565	0.172	0.862	0.965	0.989
	PackNet-SfM * [Guizilini et al., 2020a]	\mathcal{J}	$\mathcal{P}+\text{imu}$	0.075	0.384	3.293	0.114	0.938	0.984	0.995
	LiDARTouch-SAN	$\mathcal{J}+\mathcal{L}^4$	$\mathcal{P}+\mathcal{L}_4$	0.063	0.396	3.318	0.118	0.946	0.982	0.993
	LiDARTouch-NLSPN	$\mathcal{J}+\mathcal{L}^4$	$\mathcal{P}+\mathcal{L}_4$	0.053	0.336	3.013	0.106	0.959	0.987	0.994
	LiDARTouch-S2D	$\mathcal{J}+\mathcal{L}^4$	$\mathcal{P}+\mathcal{L}_4$	0.059	0.285	2.776	0.102	0.962	0.988	0.995
LiDARTouch-Monodepth2-L	$\mathcal{J}+\mathcal{L}^4$	$\mathcal{P}+\mathcal{L}_4$	0.047	0.267	2.696	0.090	0.969	<u>0.991</u>	<u>0.996</u>	
LiDARTouch-ACMNet	$\mathcal{J}+\mathcal{L}^4$	$\mathcal{P}+\mathcal{L}_4$	<u>0.044</u>	<u>0.242</u>	<u>2.504</u>	<u>0.086</u>	<u>0.974</u>	<u>0.991</u>	<u>0.996</u>	

3.6 Comparison against related works

In table 3.4, we report evaluations of the four architectures presented in section 3.3.1, trained within our LiDARTouch framework against camera-only baselines.

Self-supervised camera-only methods. First, we show that training under our framework outperforms self-supervised monocular depth estimation methods at time of submission [Godard et al., 2019; Guizilini et al., 2020a; Mahjourian et al., 2018; Wang et al., 2018; Yin and Shi, 2018; Zhou et al., 2017]. We note that contrary to other methods, ours uses few-beam LiDAR as input. Furthermore, self-supervised monocular depth estimation approaches only estimate relative depth and thus are rescaled with ground truth before evaluation. With our approach, this unrealistic and impractical rescaling step is no longer needed.

Supervised camera-only methods. We also obtain better results than monocular depth estimation models trained with ground truth and optional stereo [Fu et al., 2018; Kuznetsov et al., 2017], while not requiring either of those. While the latter does not use few-beam LiDAR as input, not requiring ground truth at train time makes our method trainable at scale on any domain.

Overall, we showed that by integrating few-beam LiDAR in the pipeline, we substantially increase performances on all metrics over other methods not using few-beam LiDAR.

We compare our LiDARTouch framework against two supervision schemes from the depth completion literature: full-supervision with ground truth (\mathcal{L}_{gt}) and self-supervision (\mathcal{L}_4 -naïve). These results are reported for the three architectures in table 3.5.

Supervised depth completion methods. Unsurprisingly, supervising the training of any of the architectures with the privileged ground-truth depth yields better results than our LiDARTouch framework. However, LiDARTouch remains very competitive, e.g., 2.504 vs.

TABLE 3.5: **Comparison against supervised and naively self-supervised depth completion schemes.** Input includes the image and the 4-beam LiDAR ($\mathcal{J}+\mathcal{L}^4$)

	Network	Superv.	Abs Rel ↓	Sq Rel ↓	RMSE ↓	$\delta < 1.25 \uparrow$
GT sup.	ACMNet	L_{gt}	0.030	0.143	2.112	0.983
	NLSPN	L_{gt}	0.044	0.214	2.617	0.971
	S2D	L_{gt}	0.035	0.152	2.271	0.979
	SAN	L_{gt}	0.037	0.172	2.491	0.976
Naive self-sup.	ACMNet	L_4	0.714	9.751	15.88	0.057
	NLSPN	L_4	4.133	268.4	51.96	0.010
	S2D	L_4	0.849	12.84	17.53	0.077
	SAN	L_4	0.426	6.226	14.148	0.243
LiDARTouch	ACMNet	$P+L_4$	0.044	0.242	2.504	0.974
	NLSPN	$P+L_4$	0.053	0.336	3.013	0.959
	S2D	$P+L_4$	0.059	0.285	2.776	0.962
	SAN	$P+L_4$	0.063	0.396	3.318	0.946

2.112 in RMSE for ACMNet while not requiring any ground truth at train time. We also investigate the impact of the density of the input LiDAR on these scores in [fig. 3.6](#). We observe that LiDARTouch is consistently close to the fully-supervised depth completion alternative when the number of layers varies.

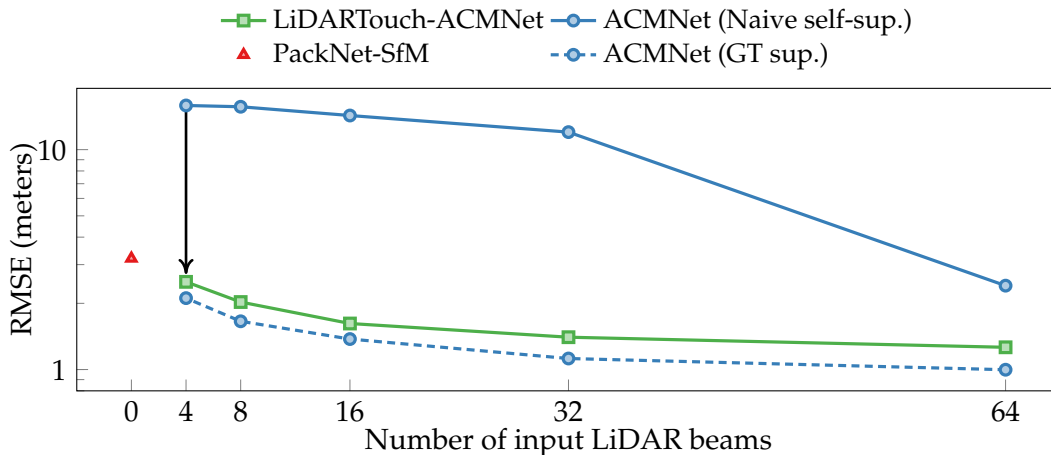


FIGURE 3.6: **Comparison of different supervision schemes for the ACMNet architecture.** In the depth-completion setting, results are highly degraded when ground-truth depth information is no longer available for supervision (blue plots, ‘GT sup.’ vs. ‘Naive self-sup.’). By combining ideas from self-supervised monocular depth estimation along with a careful integration of the LiDAR signal, we show that our self-supervised LiDARTouch framework can reach performance very close to the one offered by fully-supervised depth completion, as illustrated by the black arrow. Note that the y -axis is log-scaled.

Self-supervised depth completion method. The results in [table 3.5](#) show that the models trained with naïve 4-beam LiDAR self-supervision are unable to converge to decent results. Architectures cannot generalize from such a sparse LiDAR input as the supervisory signal is not sufficient. Moreover, in [fig. 3.6](#), we remark that the naïve self-supervision scheme makes performance plummet when the LiDAR data becomes sparser. Furthermore, for the sake of completeness, we also experiment with SAN [Guizilini et al., 2021], a recent depth completion method with similar fusion scheme to the Monodepth2-L we propose in

section 3.3.1. Overall the results of SAN in table 3.4 and table 3.5 fall within the expected range, i.e., better than camera-only methods.

3.7 Alleviating the infinite-depth problem

We now study the infinite-depth problem affecting traditional pipelines and how well does the LiDARTouch framework solve it. First, we introduce a new metric to assess the degree and the frequency to which a model dramatically overestimates the distance to cars ahead (section 3.7.1). This metric is employed for a quantitative evaluation of the problem in section 3.7.2. Besides, we also provide a qualitative analysis of the problem and the significant improvements offered by LiDARTouch (section 3.7.3).

3.7.1 Catastrophic Distance Rate (CDR) metric

Monocular image-only depth estimation methods suffer from the infinite-depth problem: vehicles with a motion close to that of the ego vehicle (in other words, with almost no relative motion) can be estimated as being infinitely far away. In the context of autonomous vehicles, such anomalies can lead to potentially dangerous outcomes. This critical weakness of image-only methods is not well reflected in the commonly-used evaluation metrics, as errors associated with these local flaws are overwhelmed by global scores aggregated at a dataset level.

This problem was qualitatively evaluated in some recent work [Casser et al., 2019a,b; Godard et al., 2019; Guizilini et al., 2020a; Zhou et al., 2017] but no precise measurement of its severity has yet been proposed. To address this issue, we define a novel quantitative metric, called the catastrophic distance rate (CDR), to assess the degree to which a model tends to make such disastrous predictions.

CDR measures the percentage of cars whose estimated distance to the ego-car is catastrophically poor in the test set. To this end, we use instance segmentation masks for all the vehicles of every image of the test set. With these vehicle instances, CDR is computed in a two-step process:

1. Instance mask filtering to keep the ones potentially concerned by the infinite-depth problem;
2. Computation of the depth error measured on these instance masks.

Instance mask filtering. For the first step of our CDR metric, we filter out irrelevant masks to only focus on vehicles typically concerned by the infinite-depth problem, i.e., first vehicle in front, unoccluded and not too far. As we use a centered frontal camera, we begin by discarding vehicles that are not in the center of the scene. We also remove cars whose instance masks are too small, considered too far from the ego vehicle. Then, to assess whether a car is occluded or not, we assume that a heavily occluded vehicle generally has a non-convex apparent shape (e.g., incised by the front vehicle) and that, on the contrary, the mask of a non-occluded car is approximately convex. The overall process is illustrated in fig. 3.7.

CDR computation. CDR estimates the percentage of instances for which the relative depth error is above a manually-defined “catastrophic” threshold τ . Within each segmentation mask M_k , indexed by $k \in \mathcal{K}$, we define the set \mathcal{V}_k of pixels that possess a ground-truth LiDAR depth measurement: $\mathcal{V}_k = \{p \mid M_k(p) > 0 \wedge D_k(p) > 0\}$. Note that, as with H_t , $D(p) = 0$ if and only if there is no LiDAR point projecting at p . In the KITTI test set,

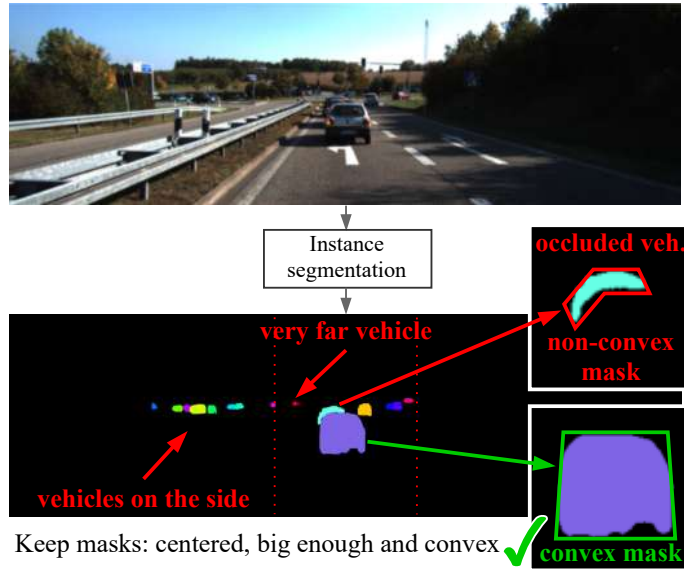


FIGURE 3.7: **Selecting vehicles to compute the CDR metric.** The aim is to extract the individual mask of the first vehicles in front of the ego-car. These are indeed vehicles affected by infinite-depth error due to a small relative motion, leading to potentially catastrophic consequences. The proposed CDR metric computes the rate of such failures over the test set.

the average size of \mathcal{V}_k is 543. The error R_k made by the model on the instance mask M_k is measured by the average signed depth error over \mathcal{V}_k :

$$R_k = \frac{1}{|\mathcal{V}_k|} \sum_{p \in \mathcal{V}_k} \frac{\hat{D}_k(p) - D_k(p)}{D_k(p)}, \quad (3.7)$$

where $|\mathcal{V}_k|$ is the cardinality of \mathcal{V}_k . Please note that no absolute value is involved in the design of R_k as we focus only on the infinite-depth problem, i.e., $\hat{D}(p) > D(p)$, when a car is predicted catastrophically further away than its true position.

By thresholding the error R_k and aggregating it over instances, we define the ‘‘Catastrophic Distance Rate’’ as:

$$\text{CDR}(\tau) = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \llbracket R_k > \tau \rrbracket, \quad (3.8)$$

with $\llbracket \cdot \rrbracket$ the Iverson bracket, $|\mathcal{K}|$ the number of instance masks and τ a user-defined threshold. For example, $\text{CDR}(\tau = 0.5) = 20\%$ indicates that the distance to front vehicles is over-estimated by more than 50% of the true distance in 20% of the cases.

3.7.2 Quantitative analysis

To verify our intuition that LiDAR self-supervision is a suitable means to mitigate the infinite-depth problem, we study three models:

- A model that does not use the LiDAR signal at all, noted ‘Monodepth w/ IMU supervision’, which heavily suffers from the infinite-depth issue;
- A model with LiDAR as input and for the PnP-estimated pose, but supervised solely with the photometric loss, noted ‘ACMNet_{PnP}^P’;

- A model trained within the LiDARTouch framework, using LiDAR for the depth network, pose estimation and self-supervision, noted ‘LiDARTouch-ACMNet’.

We plot the distribution of the CDR metric against the chosen threshold τ in [fig. 3.8](#). We observe that the more LiDAR information is integrated, the fewer catastrophic estimations occur.

Indeed, $\text{ACMNet}_{\text{PnP}}^{\text{P}}$, which uses LiDAR both in input and pose, improves over Monodepth2 but is still affected by the infinite-depth issue. We also see a clear improvement of our LiDARTouch-ACMNet over the two other models. For example, for $\tau = 0.5$, i.e., the distance of a car is overestimated by at least half, Monodepth2 has a metric score of 5.02% while $\text{ACMNet}_{\text{PnP}}^{\text{P}}$ has 0.6% and LiDARTouch-ACMNet 0.0%. Such results show that Monodepth2 predictions cannot be trusted for downstream tasks such as car detection or free space estimation that are both required by functions like automatic emergency braking, keep-lane assist or adaptive cruise control. While $\text{ACMNet}_{\text{PnP}}^{\text{P}}$ reduces the likelihood of catastrophic estimation by 8 folds for $\tau = 0.5$, 0.6% is still too high to implement in a critical system intended for wide commercial use.

Overall, a network trained with our pipeline is significantly less impacted by the infinite-depth problem and we validate our hypothesis that, during training, the LiDAR self-supervision disambiguates cars estimated too far from their real distance. Hence, our models can accurately and safely handle moving objects with no relative motion, typical of cars in fluid traffic.

3.7.3 Qualitative analysis

The three examples in [fig. 3.9](#) illustrate the improvement of our framework over the classic self-supervised camera-only pipeline. On the leftmost column, we observe a typical ‘hole’ in the depth map where Monodepth2 with IMU supervision estimates a vehicle three times more distant than in reality. In contrast to our model without such holes.

In addition to [fig. 3.9](#), we provide some qualitative analyses where we show the depth maps obtained for different frameworks in [fig. 3.10](#). First, we observe better overall depth maps with LiDARTouch-ACMNet than with Monodepth2. For example, we better estimate the two moving cyclists in [fig. 3.10a](#) as well as the fine tree trunks in [fig. 3.10c](#).

As expected, the fully-supervised method ACMNet (GT-sup.) delivers the best-qualitative depth maps, as it leverages privileged ground-truth LiDAR depth during training. However, we observe that self-supervised approaches (Monodepth2 and LiDARTouch-ACMNet) better estimate areas near the top of the scene. This can be explained as LiDAR points are

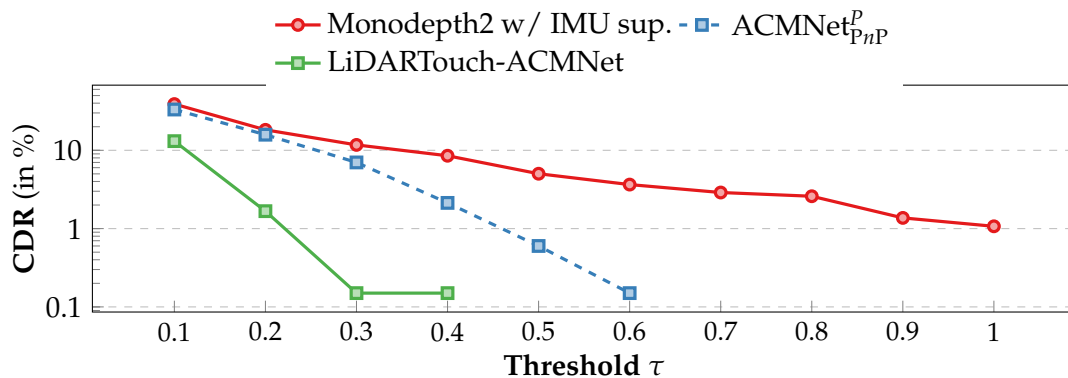


FIGURE 3.8: Plot of the CDR metric for various thresholds τ . The y -axis is log-scaled.

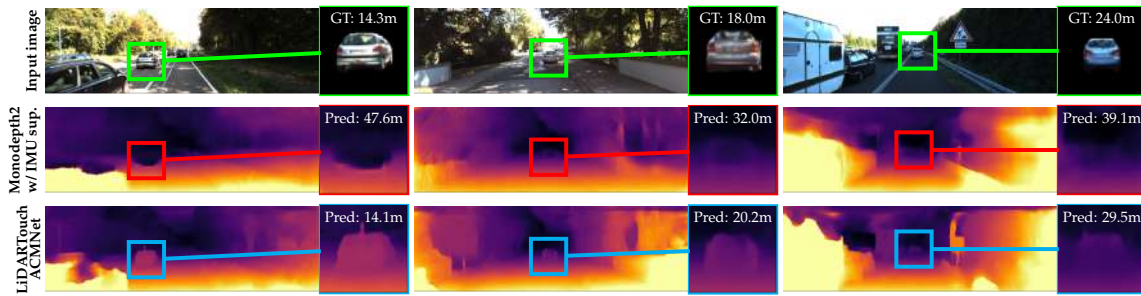


FIGURE 3.9: **Mitigation of the infinite-depth problem.** Self-supervised image-only approaches tend to predict objects with no relative-motion at an infinite depth, as indicated by the hole in the depth close-up (red). In contrast, our LiDARTouch framework estimates the depth of these vehicles, as shown in the green close-up. Note that for the example in the middle, we verified that no LiDAR measurement falls on the car. This shows that our training framework can generalize well to cases where no LIDAR is available on critical moving vehicles.

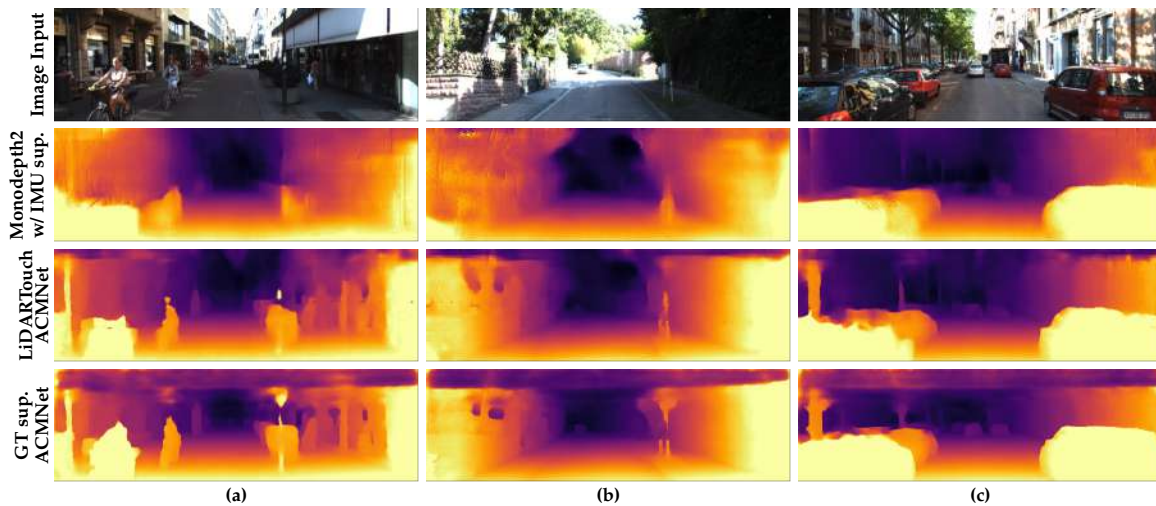


FIGURE 3.10: **Qualitative comparison of LiDARTouch with other existing frameworks.** Monodepth2 is trained with IMU supervision. The model trained with GT supervision gives sharper depth estimates, but struggles in regions where GT signal is not available (e.g., top of the scene).

absent from regions above the road, which hinders ACMNet (GT-sup.) prediction in these regions due to the lack of supervisory signal it uses (last row in [fig. 3.10](#)).

Despite the successful integration of LiDAR in LiDARTouch, we note that some local depth estimation artifacts still occur, similar to the maps obtained from self-supervised depth estimation methods. Typically, this concerns distorted, reflective and color-saturated regions because the photometric reconstruction loss assumes Lambertian surfaces (cars in [fig. 3.10c](#)). Our model may also produce blurry depth predictions for small or thin objects, such as traffic signs (Figures [3.10a](#) and [3.10b](#)).

3.8 Implementation details

Training. We train all our models for 30 epochs using the Adam optimizer [[Kingma and Ba, 2015](#)] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The initial learning rate is set to $1e-4$ and divided by two halfway through training.

In all training pipelines, following common practice [Godard et al., 2019, 2017; Guizilini et al., 2020a], we add an edge-aware smoothing regularization loss to encourage the predicted depth map \hat{D}_t to be locally smooth while taking into account sharp boundaries:

$$L_{\text{smooth}} = |\partial_x \hat{D}_t| e^{-|\partial_x I_t|} + |\partial_y \hat{D}_t| e^{-|\partial_y I_t|}, \quad (3.9)$$

with the index p over pixels omitted for clarity.

Monodepth2 extension. Our Monodepth2-L architecture is similar to Monodepth2 at the difference that we use a second ResNet-18 encoder specifically for the LiDAR modality. We only remove the first batch-normalization layer of the LiDAR ResNet, as using it would imply the computation of ineffective statistics given that the LiDAR input mostly contains zeros (encoding measurement absence).

Pose estimation. To solve the PnP problem, we use an open-source implementation of PnP methods with RANSAC from the OpenCV library [Bradski, 2000]. We use 100 iterations and a reprojection error threshold of 2. Even after RANSAC, the remaining outliers are numerous enough to hinder training. Therefore, we remove the relative pose estimates for which the translation magnitude $\|\hat{t}\|$ is too large. In effect, we first compute the median value of translation magnitude for each relative pose of the train set. Then, we remove all examples that are too far-off the median. When using a pose network, we follow [Godard et al., 2019] and use a ResNet-18 taking two images in input and outputting the parameters of $\hat{P}_{t \rightarrow s}$, the rigid transformation between the two views.

Evaluation after rescaling. Baselines and models from prior works that only provide relative-depth maps have their predictions rescaled so that they have the same mean compared to the ground truth against which they are evaluated. This is mentioned as ‘*gt rescaled*’ in table 3.4. For methods that directly produce metric depth maps, like ours, we do not apply this post-processing procedure and depth maps are kept at the originally-predicted scale.

CDR Metric. To compute results with our CDR metric, we first extract instance masks with EfficientPS [Mohan and Valada, 2021]. Among these masks, we want to focus only on those of close-by, non-occluded vehicles, i.e., first vehicles in front of the ego-car. These vehicles are particularly prone to infinite-depth mistakes, with safety-critical consequences when it happens. To do this selection, vehicles that are not in front of the ego-car are discarded, as measured by not belonging to the central band of the scene (size is 20% of the image width) captured by the front camera. Vehicle having instance masks calculated with fewer than 20 pixels are considered too far from the ego vehicle. Then, to assess whether a car is occluded or not, we assume that a heavily occluded vehicle generally has a non-convex shape (e.g., incised by the front vehicle) and that, on the contrary, the mask of a non-occluded car is approximately convex. Accordingly, we first smooth segmentation masks and fill noisy areas where the intensity changes rapidly (e.g., edges, small holes from the wheels) by applying a morphological *dilation* operator. We use a square kernel of size 10 and 4 iterations for this operation. The masks now being smoothed, we then approach their shape by a polygon from which we can tell if they are convex or not. To approximate each pixel blob by a polygon, we use the Douglas–Peucker algorithm [Douglas and Peucker, 1973]. The algorithm ensures the fit of the approximated polygon with an accuracy parameter dependent of the pixel blob size. After this first filtering step, 657 valid masks remain out of the 4460 vehicle masks of the KITTI test split.

Extracting 4 beams from 64-beam point clouds. In the KITTI dataset, the LiDAR data in a frame is provided as a unique point cloud, that is, a set of (x, y, z) coordinates, without the beam indexes, i.e., which of the 64 lasers has been used for each measurement. We needed to recover this information for our experiments. Fortunately, in KITTI the points

are recorded in an orderly manner. The points of one beam follow the points of another in the direction of laser rotation (counter-clockwise). This means that, inside the data stream of a same frame, each rotation completion indicates a change of beam. More precisely, the coordinate basis of the LiDAR is oriented with x : positive forward and y : positive to the left of the car. Then we can compute the horizontal angle in radian of each point with:

$$\phi = \arctan2(y, x). \quad (3.10)$$

We use the 2-argument arctangent instead of classic arctangent, $\arctan(y/x)$, as the latter cannot distinguish between diametrically opposite directions. Then, by computing the horizontal angle (azimuth) of each point, we can separate data for each beam by detecting when ϕ changes from 360° to 0° in the stream of points. This way, we have access to the ring index for each LiDAR point and can, thus, freely sparsify the LiDAR data.

3.9 Conclusion

In this chapter, we proposed LiDARTouch, a novel self-supervised framework for depth estimation with a monocular camera and a few-beam LiDAR.

3.9.1 Summary of contributions

Alleviate monocular ambiguities. While extremely sparse, we show that the LiDAR signal can be leveraged to alleviate the *ambiguity of scale* and the *infinite-depth* issues that monocular depth estimation methods face.

Performant, cheap, self-supervised. The LiDARTouch framework can reach competitive performances with respect to fully-supervised depth completion methods while being significantly cheaper and not requiring dense annotation. Thus, our method can be trained for accurate and metric depth estimation on any domain with no modification from the raw sensors signal only, taking full advantage of the data acquired by a vehicle fleet.

Flexibility of the framework. We have validated the influence of the LiDAR integration at three complementary levels of the self-supervised learning scheme, across five different architectures, highlighting the robustness and the adaptability of our learning system to diverse deep architectures.

New metric. Along with the new CDR metric to measure the infinite-depth problem, and the associated source code, we hope to enable further research on the task of monocular depth prediction with minimal LiDAR input, typical of real-world assisted/automated driving systems.

3.9.2 Perspectives

Extension to full surround. This work mostly focus on a “frontal camera” setup, i.e., only one camera facing forward, while L3+ automatized vehicles are typically equipped with multiple cameras organized such that a full surround observation of the surrounding is possible. The recent work of [Guizilini et al. \[2022a\]](#) takes advantage of such sensor setups by projecting views between cameras in addition to projecting views across time, leading to a more precise depth estimation that is also coherent across cameras. Such approach could directly be adapted in the LiDARTouch framework to handle full surround camera setups.

Better photometric loss. Another aspect that can be improved is the photometric reconstruction objective. The SSIM+L1 loss used in this work assumes uniform lighting conditions and Lambertian surfaces (i.e., a surface that scatters incident illumination equally in all directions), making reflective and transparent surfaces difficult to estimate. Likewise, textureless regions and surfaces with repeated patterns can be difficult to learn on. To address these problems, [Shu et al. \[2020\]](#) propose to define the reconstruction loss in a learned feature space instead of the RGB space (i.e., a feature map is projected between views instead of RGB values). This feature space is learned to favour discriminative features, making the reconstruction loss easier to optimize and improving the depth quality on fine details and textureless regions.

Improving robustness to visual impediments. Alternatively, [Kaushik et al. \[2021\]](#) propose a consistency loss on the depth under strong augmentation (e.g., brightness, jitter, gamma, saturation, Gaussian noise). The inductive bias introduced that the 3D geometry of the scene is independent of visual variations like lightning, colours, and visual noise. This concept could be extended to visual impediments typical of conditions that can affect the perception capabilities of automatized vehicles, i.e., synthesized glares or droplets of water.

Chapter 4

Latents and Rays for an Implicit Scene Representation

As presented in [section 2.3.3](#), recent works in autonomous driving have widely adopted the Bird’s-Eye-View (BEV) semantic map as an intermediate representation of the world. Nonetheless, online prediction of these BEV maps involves non-trivial operations such as multi-camera data extraction as well as fusion and projection into a common top-view grid. This is usually done with error-prone geometric operations (e.g., homography or back-projection from monocular depth estimation) or expensive direct dense mapping between image pixels and pixels in BEV (e.g., with MLP or attention).

In this chapter, we present an efficient and general encoder-decoder, transformer-based model for vehicle semantic segmentation from multiple cameras. This method, called ‘LaRa’ has been published in 2022, in the scientific conference CoRL [[Bartoccioni et al., 2022](#)]. Our approach uses a system of cross-attention to aggregate information over multiple sensors into a compact, yet rich, collection of latent representations. These latent representations, after being processed by a series of self-attention blocks, are then reprojected with a second cross-attention in the BEV space. We demonstrate that our model outperforms the best previous works, at the time of submission, on nuScenes. The code and trained models are available at <https://github.com/valeoai/LaRa>.

4.1 Introduction

To plan and drive safely, autonomous cars need accurate 360-degree perception and understanding of their surroundings from multiple and diverse sensors, e.g., cameras, RADARs, and LiDARs. Most of the established approaches tardily aggregate independent predictions from each sensor [[Liu et al., 2020b](#); [Roddick et al., 2019](#); [Wang et al., 2021](#)]. Such a late fusion strategy has limitations for reasoning globally at the scene level and does not take advantage of the available prior geometric knowledge that links sensors. Alternatively, the bird’s-eye-view’s (BEV) representational space, a.k.a. top-view occupancy grid, recently gained considerable interest within the community. BEV appears as a suitable and natural space to fuse multiple views [[Hu et al., 2021](#); [Phillion and Fidler, 2020](#)] or sensor modalities [[Bai et al., 2022](#); [Hendy et al., 2020](#)] and to capture semantic, geometric, and dynamic information. Besides, it is a widely adopted choice for downstream driving tasks including motion forecasting [[Caesar et al., 2020](#); [Chang et al., 2019](#); [Ettinger et al., 2021](#); [Hu et al., 2021](#)] and planning [[Caesar et al., 2021](#); [Casas et al., 2021](#); [Chitta et al., 2021](#); [Zeng et al., 2019](#)].

In this thesis, we focus on BEV perception from multiple cameras. The online estimation of BEV representations is usually done by: (i) imposing strong geometric priors such as a flat world [[Reiher et al., 2020](#)] or correspondence between pixel columns and BEV rays [[Roddick and Cipolla, 2020](#)], (ii) predicting depth probability distribution over pixels

to lift from 2D to 3D and project back in BEV [Hu et al., 2021; Phillion and Fidler, 2020], a system subject to compounding errors, or, (iii) learning a costly dense mapping between multi-camera features and the BEV grid pixels [Zhou and Krähenbühl, 2022].

Here, we depart from these dominant strategies and introduce ‘LaRa’, a novel transformer-based model for vehicle segmentation from multiple cameras. In contrast to prior works, we propose to use a latent ‘internal representation’ instantiated as a collection of vectors. Fusing multiple views into a compact latent space comes with several benefits. First, it provides an explicit control on the memory and computation footprint of the model, instead of the quadratic scaling of the full mapping between multi-camera features and the BEV grid pixels [Zhou and Krähenbühl, 2022]. By design, the number of latents that we use is much smaller compared to the spatial resolution of the BEV grid, enabling a highly-efficient aggregation of information at the latent-level while exploiting spatial cues within and across camera views. Moreover, we also hypothesize that discarding error-prone modules in the pipeline such as depth estimation [Hu et al., 2021; Phillion and Fidler, 2020] can boost model accuracy and robustness. Finally, we can directly predict at the full-scale BEV resolution bypassing noisy upsampling operations. This is infeasible, within a reasonable computational budget, for prior works restricted to coarser BEV grids as they map densely between all the image and BEV grid pixels [Zhou and Krähenbühl, 2022]. Besides, as an orthogonal contribution, we augment input features with ray embeddings that encode geometric relationships within and across images. We show that such spatial embeddings, encoding prior geometric knowledge, help guide the cross-attention between input features and the latent vectors.

Our approach is extensively validated against prior works on the nuScenes [Caesar et al., 2020] dataset. We significantly improve the performance on the vehicle segmentation task, outperforming recent high-performing models [Phillion and Fidler, 2020; Zhou and Krähenbühl, 2022]. Moreover, we show interesting properties of our cross-attention, which naturally stitches multiple cameras together. We also perform several ablation and sensitivity studies of our architecture with respect to hyper-parameters changes. Overall, LaRa is a novel model that learns the mapping from camera views to bird’s-eye-view for the task of vehicle semantic segmentation. In summary, our contributions are as follows:

- We encode multiple views into a compact latent space that enables precise control on the model’s memory and computation footprint, decoupled from the input size and output resolution.
- We augment semantic features with spatial embeddings derived from cameras’ calibration parameters and show that it strongly helps the model learn to stitch multiple views together.
- Our architectural contributions are validated on nuScenes where we reach new SOTA results.

4.2 Related work

4.2.1 BEV semantic segmentation

Models for BEV segmentation are typically structured in two parts. They first extract features of each camera and then project them into a common top-view grid, called the bird’s-eye-view. There are different strategies for this projection, which can be grouped into the following categories.

IPM-based. Inverse perspective mapping (IPM) defines the correspondence between the camera and the ground planes as a homography matrix. IPM makes strong assumptions

that the world is planar and the cameras' horizontal axes are parallel to the ground. Early works [Bertozzi et al., 1998; Sengupta et al., 2012] apply it directly to raw camera pixels or features. This approach suffers from blurring and stretching artifacts for distant objects (as they have fewer pixels in the camera view) and objects with a height (as they violate the planar world assumption). To alleviate these shortcomings, a generative adversarial network [Zhu et al., 2018] or training a BEV decoder with synthetic ground-truth [Reiher et al., 2020] has been used to refine the IPM projection.

'Lift-splat'-based: guiding with depth. Using depth information to lift features from 2D to 3D and then 'splatting' them in BEV space recently gained popularity for its effectiveness and sound geometric definition. Among the formulations of depth estimation for BEV projection [Hu et al., 2021; Ng et al., 2020; Phillion and Fidler, 2020; Roddick et al., 2019; Srikanth et al., 2019], estimating depth probabilities along camera rays appears to perform the best [Hu et al., 2021; Phillion and Fidler, 2020]. However, depth being the most influential factor [Simonelli et al., 2021], such a strategy is subject to compounding errors. Inaccuracies in the depth prediction will propagate into the BEV features, which themselves can be degraded.

Implicitly learned with dense networks. An alternative to explicit geometric projection is to learn the mapping from data. For instance, VPN [Pan et al., 2020] uses an MLP to make a dense correspondence between pixels in the camera views and BEV. This kind of method relies on expensive operations and does not use readily available spatial information given by the calibrated camera rig capturing the images. The BEV projection must be entirely learned, and as it is determined by training data, it can hardly apply to new settings with slightly different camera calibrations. Alternatively, PON [Roddick and Cipolla, 2020] builds on the observation that a column in the camera image contains all the information of the corresponding ray in BEV: it first encodes each column into a feature vector, which is then decoded into a ray along the depth dimension. However, this relies on two implicit assumptions: (i) the camera follows a pinhole projective model, and (ii) it is horizontally aligned with the ground plane.

Implicitly learned with transformer architectures. The attention system at the core of transformer architectures [Carion et al., 2020; Dosovitskiy et al., 2021; Jaegle et al., 2022; Vaswani et al., 2017] allows learning of long-range dependencies and correspondences explicitly. These architectures have recently been employed for the BEV semantic segmentation task, yielding among the best-performing methods [Gong et al., 2022; Li et al., 2022b; Zhou and Krähenbühl, 2022]. Nonetheless, a direct cross-attention [Vaswani et al., 2017] between camera images and the BEV grid is computationally expensive. BEVFormer [Li et al., 2022b] alleviates this issue by only cross-attending BEV pixels with cameras in which the BEV pixel is visible and by replacing the heavier multi-head attention [Vaswani et al., 2017] with deformable attention [Zhu et al., 2021]. CVT [Zhou and Krähenbühl, 2022] keeps the vanilla multi-head cross-attention [Vaswani et al., 2017] but applies it between low-resolution camera feature maps and a small BEV grid which is then upsampled to reach the final resolution. GitNet [Gong et al., 2022] restrains the cross-attentions to column-ray pairs making the same original implicit assumptions as PON [Roddick and Cipolla, 2020]. Our proposed model LaRa belongs to this category as it learns the BEV representation with a transformer architecture. On the other hand, our attention scheme does not impose strong geometric assumptions while still being efficient enough to attend to a full-resolution BEV grid.

4.2.2 Incorporating geometric priors in Transformers

Since transformer architectures are permutation-invariant, spatial relationships between image regions are lost if no precautions are taken. A standard practice to retain this spatial knowledge is to add a positional embedding to the input of attention layers [Vaswani et al., 2017]. A popular approach is to encode the position of pixels with sine and cosine functions of varying frequencies [Carion et al., 2020; Jaegle et al., 2022; Vaswani et al., 2017] applied over the horizontal and vertical axes. An alternative solution to induce spatial awareness in the model is to concatenate x, y positions to feature maps fed to convolutional layers [Liu et al., 2018].

Related to our ray embedding proposition, recent works [Yifan et al., 2022; Zhou and Krähenbühl, 2022] embed the parameters of the calibrated cameras in the image features, improving training efficiency and segmentation performance. Similar to LaRa, IIB [Yifan et al., 2022] also encodes the camera center and ray direction in the input feature sequence, but it addresses the task of depth estimation on image pairs in an indoor environment. Furthermore, Yifan et al. [2022] embed the origin and direction of rays into Fourier features, which can become memory intensive depending on the number of frequency bands and also introduces additional hyper-parameters to tune. CVT [Zhou and Krähenbühl, 2022] adds up a ray direction embedding to the input feature sequence, but, differently from ours, uses the camera center embedding in the BEV query. This requires a BEV query and ‘cross-view attention’ operation per camera, increasing the memory and computational footprint, thus limiting the maximum resolution of the BEV query.

4.3 LaRa: Our Latents and Rays Model

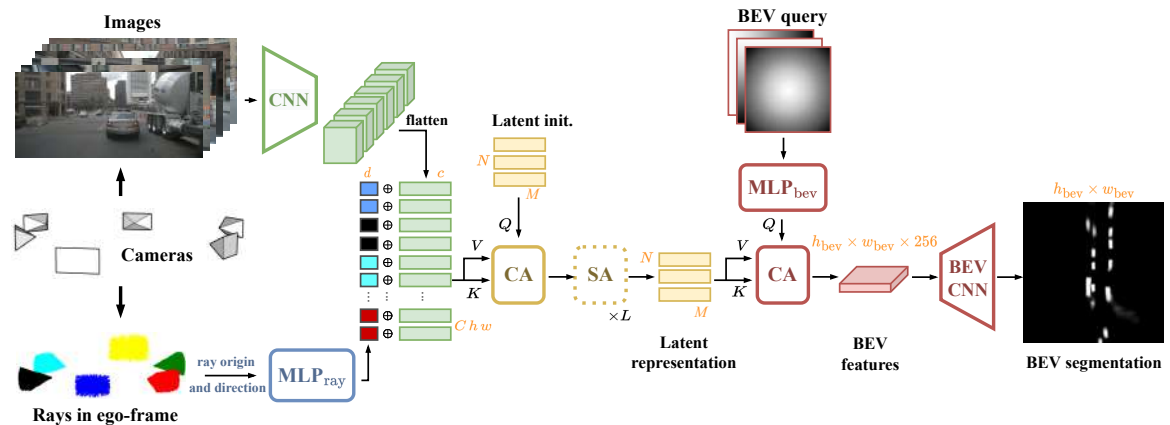


FIGURE 4.1: **LaRa overview.** Semantic features (green) are extracted from the images with a shared CNN and are concatenated with ray embeddings (multi-coloured) that inform about geometric information to spatially relate pixels within and across cameras. This representation is then fused into a compact latent representation through one cross-attention (CA) and L self-attention (SA) layers (yellow). The final BEV map is obtained by querying the latent representation with a cross-attention and then refined with BEV CNN (red). \oplus denotes concatenation. The orange letters indicate tensor dimensions. K , Q , and V are the *Key*, *Query*, and *Value* of the cross-attentions.

Given multiple cameras observing the scene, our goal is to estimate a binary occupancy grid [Elfes, 1990] $\hat{y} \in \{0, 1\}^{h_{\text{bev}} \times w_{\text{bev}}}$ of size $h_{\text{bev}} \times w_{\text{bev}} \in \mathbb{N}^2$ for vehicles in the surroundings of the ego car. We propose ‘LaRa’ a transformer-based architecture to efficiently aggregate information gathered from multiple cameras into a compact latent representation

before expanding back into the BEV space. Besides, as we believe that the geometric relationship between cameras should guide the fusion across each camera view, we propose to augment each pixel with the geometry of the ray that captured it. The LaRa architecture is illustrated in [fig. 4.1](#).

4.3.1 Input modelling with geometric priors

We consider C cameras described by $(I_k, \mathcal{K}_k, \mathcal{R}_k, t_k)_{k=1}^C$, with $I_k \in \mathbb{R}^{H \times W \times 3}$ the image produced by camera k , $\mathcal{K}_k \in \mathbb{R}^{3 \times 3}$ the intrinsics, $\mathcal{R}_k \in \mathbb{R}^{3 \times 3}$ and $t_k \in \mathbb{R}^3$ the extrinsic rotation and translation respectively. From these inputs, two complementary types of information are extracted: visual information from raw images and geometric cues from the camera calibration parameters.

Visual information from raw images. A shared image-encoder E extracts feature maps for each image $F_k = E(I_k) \in \mathbb{R}^{h \times w \times c}$. Following [Hu et al., 2021; Philion and Fidler, 2020], we instantiate E with a pretrained EfficientNet [Tan and Le, 2019] backbone to produce the multi-camera features. These spatial feature maps in $\mathbb{R}^{C \times h \times w \times c}$ are then rearranged as a sequence of feature vectors, in $\mathbb{R}^{(Chw) \times c}$.

Leveraging geometric priors. To enrich camera features with geometric priors, commonly used sine and cosine spatial embeddings [Carion et al., 2020; Jaegle et al., 2022; Vaswani et al., 2017] are ambiguous in the presence of multiple cameras. A straightforward solution would be to use camera-dependant learnable embeddings in addition to the Fourier embeddings to disambiguate between cameras. However, in our setting, we argue that the geometric relationship between cameras, which is defined by the structure of the camera rig, is crucial to guide the fusion of the views. This motivates our choice to leverage the cameras' extrinsics and intrinsics to encode the position and orientation of each pixel in the vehicle ego-frame.

More precisely, we encode the camera calibration parameters by constructing the viewing ray for each pixel of the cameras. Given a pixel coordinate $x \in \mathbb{R}^2$ within a camera image I_k , the direction $d_k(x) \in \mathbb{R}^3$ of the ray that captured x is computed with:

$$d_k(x) = \mathcal{R}_k^{-1} \mathcal{K}_k^{-1} \tilde{x}, \quad (4.1)$$

where \tilde{x} are the homogeneous coordinates of x , and $d_k(x)$ is expressed in ego-coordinates. The origin of the ray $d_k(x)$ is the camera center given by t_k .

Then, to fully describe the position and the orientation of the ray that captured pixel x , we use the embedding $ray_k(x) \in \mathbb{R}^d$ computed as follows:

$$ray_k(x) = \text{MLP}_{\text{ray}}(t_k \oplus d_k(x)), \quad (4.2)$$

where \oplus is a concatenation operation and MLP_{ray} a 2-layer MLP with GELU activations [Hendrycks and Gimpel, 2016]. Note that the intrinsics are scaled according to the difference in resolution between I_k and F_k . As shown in [fig. 4.1](#), the final input vector sequence, in $\mathbb{R}^{(Chw) \times (d+c)}$, is produced by concatenating each of the C hw feature vectors $F_k(x) \in \mathbb{R}^c$ with its geometric embedding $ray_k(x) \in \mathbb{R}^d$.

4.3.2 Building latent representations and deep fusion

To control the computational and memory footprint of the image-to-BEV block, we leverage findings from general-purpose architectures [Jaegle et al., 2022] and propose to use an intermediate fixed-sized latent space instead of learning the quadratic all-to-all correspondence between multi-camera features and BEV space [Zhou and Krähenbühl, 2022]. Formally, the visual representations F_k from all cameras, along with their corresponding

geometric embeddings ray_k , are compressed by cross-attention [Vaswani et al., 2017] into a collection of N learnable latent vectors of dimension $M \in \mathbb{N}$ and processed by a series of L self-attention blocks [Vaswani et al., 2017] (see yellow elements in fig. 4.1). We stress that $N \ll Chw$, which enables to fuse and process efficiently the visual information coming from all the cameras, regardless of the input feature resolution or the number of cameras. Thanks to latent-based querying, this formulation decouples the network’s deep multi-view processing from the input and output resolution. Our architecture can thus take advantage of the full resolution of the BEV grid.

4.3.3 Generating BEV output from latents

The final step is to decode the binary segmentation prediction $\hat{y} \in \{0, 1\}^{h_{\text{bev}} \times w_{\text{bev}}}$ from the latent space. In practice, the latent vectors are cross-attended [Vaswani et al., 2017] with a BEV ‘query’ grid $Q \in \mathbb{R}^{h_{\text{bev}} \times w_{\text{bev}} \times d_{\text{bev}}}$ at the final prediction resolution, with $d_{\text{bev}} \in \mathbb{N}$ a hyper-parameter (illustrated by the red blocks in fig. 4.1). Each element of the query grid is a feature vector encoding the spatial position in the bird’s-eye-view which specifies what information the cross-attention would extract from the latent representations. This last cross-attention yields a feature map in BEV space, in dimension $h_{\text{bev}} \times w_{\text{bev}} \times 256$, that is further refined with a small convolutional encoder-decoder U-Net (‘BEV CNN’ in fig. 4.1) to finally predict the binary bird’s-eye-view semantic map $\hat{y} \in \{0, 1\}^{h_{\text{bev}} \times w_{\text{bev}} \times 1}$.

Specifically, we consider a combination of two types of queries: normalized coordinates in the BEV space and radial distance. Normalized coordinates encode ego-centered normalized coordinates of the BEV plane. They are obtained with:

$$Q_{\text{coords}}[i, j] = \left(\frac{2i}{h_{\text{bev}} - 1} - 1, \frac{2j}{w_{\text{bev}} - 1} - 1 \right), \forall i, j \in \{0, \dots, h_{\text{bev}} - 1\} \times \{0, \dots, w_{\text{bev}} - 1\}. \quad (4.3)$$

Normalized radial distances are simply Euclidean distances of pixels w.r.t. the origin:

$$Q_{\text{radial}}[i, j] = \sqrt{Q_{\text{coords}}[i, j]_i^2 + Q_{\text{coords}}[i, j]_j^2}. \quad (4.4)$$

While the network could produce a similar embedding from Q_{coords} using MLP_{bev} , we find that introducing these radial embeddings along Q_{coords} empirically improves results. Moreover, this query decoding choice compares favorably against more classical Fourier embeddings [Jaegle et al., 2022; Vaswani et al., 2017; Yifan et al., 2022] and learned query embeddings [Carion et al., 2020; Vaswani et al., 2017], as shown in table 4.3.

4.4 Experiments

4.4.1 Evaluation details

Dataset. We conduct experiments on the nuScenes dataset [Caesar et al., 2020], which contains 34k annotated sets of frames captured by $C=6$ synchronized cameras covering the 360° field of view around the ego vehicle. The extrinsics and intrinsics calibration parameters are given for all cameras in every scene. Raw annotations come in the form of 3D bounding boxes that are simply rendered in the discretized top-down view of the scenes to form the ground-truth for our binary semantic segmentation task.

Precise settings for training and validation. With no established benchmarks to precisely compare model’s performances, there are almost as many settings as there are previous works. Differences are found at three distinct levels:

- The **resolution** of the output grid where two main settings have been used: a grid of $100\text{m} \times 50\text{m}$ at a 25cm resolution [Pan et al., 2020; Roddick and Cipolla, 2020; Saha et al., 2021; Zhou and Krähenbühl, 2022] and a grid of $100\text{m} \times 100\text{m}$ at a 50cm resolution [Phillion and Fidler, 2020; Zhou and Krähenbühl, 2022]. These settings are respectively referred as ‘Setting 1’ ($h_{\text{bev}} \times w_{\text{bev}} = 400 \times 200$) and ‘Setting 2’ ($h_{\text{bev}} \times w_{\text{bev}} = 200 \times 200$) and they are clearly specified when we present our results.
- The considered **classes**. There are slight differences in the classes used to train and evaluate the model. For instance, some models are trained with a multi-class objective to simultaneously segment objects such as `cars`, `pedestrian` or `cones` [Pan et al., 2020; Roddick and Cipolla, 2020; Saha et al., 2021]. Some others only train and evaluate in a binary semantic segmentation setting on a meta-class `vehicles` which includes `cars`, `bicycles`, `trucks`, etc. [Phillion and Fidler, 2020; Zhou and Krähenbühl, 2022]. Some works also use *instance* segmentation information to train their model where the centers of each distinct vehicle is known at train time [Hu et al., 2021]. In our experiments, we place ourselves in the *binary semantic segmentation* setting of the meta-class `vehicles`. This choice is made to have fair and consistent comparisons with our baselines [Phillion and Fidler, 2020; Zhou and Krähenbühl, 2022], however, it should be noted that our model is not constrained to this setting.
- The levels of **visibility** of objects. Objects selected as ground truth, both for training and evaluating the model, differ in terms of their levels of visibility. Three options have been considered: objects that are in line-of-sight with the ego car’s LiDAR [Roddick and Cipolla, 2020], or objects with a nuScenes visibility above a defined threshold, either 0% [Phillion and Fidler, 2020] or 40% [Zhou and Krähenbühl, 2022]. When amenable, we clearly indicate the level of visibility used in our experiments.

In all the settings we considered, models are evaluated with the Intersection-over-Union (IoU) metric.

Training and implementation details. We train our model by optimizing the Binary Cross Entropy with our predicted soft segmentation maps and the binary ground-truth. Images are processed at resolution 224×480 . We use the AdamW [Loshchilov and Hutter, 2019] optimizer with a constant learning rate of $5e-4$ and a weight decay of $1e-7$. We train our model on 4 Tesla V100 16GB GPUs with a total batch size of 8 for 30 epochs. Training takes on average 11 hours. We use an EfficientNet-B4 [Tan and Le, 2019] with an output stride of 8 as our CNN image encoder. For the BEV CNN we follow Phillion and Fidler [2020].

Following common practice [Hu et al., 2021; Phillion and Fidler, 2020; Zhou and Krähenbühl, 2022], we employ an EfficientNet [Tan and Le, 2019] as our CNN image encoder E . In particular, we use an EfficientNet-B4 [Tan and Le, 2019] with an output stride of 8. It extracts feature maps for each image $F_k = E(I_k) \in \mathbb{R}^{h \times w \times c}$. In practice, $h = 224/8 = 28$, $w = 480/8 = 60$ and we define $c = 128$.

For the BEV CNN, we follow Phillion and Fidler [2020] and use an encoder-decoder architecture with a ResNet-18 [He et al., 2016] as backbone. It produces features at three levels of resolutions (1:1, 1:2 and 1:8), which are progressively upsampled back to the input resolution with bilinear interpolation (first $\times 4$ for the 1:8th scale then $\times 2$ for the 1:2th). Skip connections are used between encoder and decoder stages of the same resolution.

Both MLP_{ray} and MLP_{bev} are 2-layer MLPs producing 128-dimensional features. Each consists of two linear transformations with a GELU [Hendrycks and Gimpel, 2016] activation function:

$$\text{MLP}(x) = W_2 \text{GELU}(W_1 x + b_1) + b_2. \quad (4.5)$$

Following Jaegle et al. [2021], the latent vectors are randomly initialized using a truncated normal distribution with mean 0, standard deviation 0.02, and truncation bounds $[-2, 2]$.

Details on attention modules. Following the original formulation and notations [Vaswani et al., 2017], the attention operation is defined as:

$$\text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_K}}\right)V \quad (4.6)$$

with its multi-headed extension:

$$\begin{aligned} \text{MultiheadAttn}(Q, K, V) &= \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \\ \text{where head}_i &= \text{Attn}(QW_i^Q, KW_i^K, VW_i^V). \end{aligned} \quad (4.7)$$

with d_q, d_v, d_k the dimensions of Q, K and V . In practice, we use d_{model} , a hyperparameter, to define the dimension of the queries, keys and values for the inner attention (eq. (4.6)) as well as h the number of attention heads. More precisely, we linearly project queries, keys and values h times with different projections, each with dimension $d_{\text{emb}} = d_{\text{model}}/h$. The learnable projection matrices of each head are defined as $W_i^Q \in \mathbb{R}^{d_q \times d_{\text{emb}}}$, $W_i^K \in \mathbb{R}^{d_k \times d_{\text{emb}}}$, $W_i^V \in \mathbb{R}^{d_v \times d_{\text{emb}}}$ and $W_i^O \in \mathbb{R}^{h \cdot d_{\text{emb}} \times d_v}$.

Our architecture integrates three attention modules [Vaswani et al., 2017]: (i) a cross-attention between latent vectors and input features; (ii) a sequence of self-attention on the latent vectors; (iii) a cross-attention between BEV query and latent vectors. More precisely, and with a slight abuse of notation:

Latent-Input cross-attention (32 heads)

$$\begin{aligned} \text{latents} &:= \text{MultiheadAttn}(\text{LN}(\text{latents}), \text{LN}(\text{input}), \text{LN}(\text{input})) + \text{latents} \\ \text{latents} &:= \text{MLP}(\text{LN}(\text{latents})) + \text{latents} \end{aligned} \quad (4.8)$$

Latent self-attention (16 heads)

$$\begin{aligned} \text{latents} &:= \text{MultiheadAttn}(\text{LN}(\text{latents}), \text{LN}(\text{latents}), \text{LN}(\text{latents})) + \text{latents} \\ \text{latents} &:= \text{MLP}(\text{LN}(\text{latents})) + \text{latents} \end{aligned} \quad (4.9)$$

BEV query-Latent cross-attention (16 heads)

$$\begin{aligned} \text{output} &:= \text{MultiheadAttn}(\text{LN}(\text{BEVquery}), \text{LN}(\text{latents}), \text{LN}(\text{latents})) \\ \text{output} &:= \text{MLP}(\text{LN}(\text{output})) + \text{output} \end{aligned} \quad (4.10)$$

Where LN is a layer normalization [Ba et al., 2016]. In particular, the cross-attention between BEV query and latent vectors is not residual. Since the query is made of coordinates, imposing the network to predict segmentation as residual of coordinates does not make sense. The exact specification of other modules are available in our code.

4.4.2 Comparison with previous works

In table 4.1, we compare the IoU performances of LaRa against two baselines Lift-Splat [Philion and Fidler, 2020] and CVT [Zhou and Krähenbühl, 2022] on vehicle BEV segmentation in their respective training and evaluation setups. In all cases, we improve results by a significant margin. More precisely, we improve by 10% compared to Lift-Splat in their settings, by 10% and 8% compared to CVT respectively in Setting 1 and Setting 2. This suggests that our model can better extract the geometric and visual information from all

TABLE 4.1: **Intersection-over-Union (IoU) for vehicle segmentation on nuScenes.** ‘Setting 1’ refers to a $100\text{m}\times 50\text{m}$ grid with a 25cm resolution and ‘Setting 2’ to a $100\text{m}\times 100\text{m}$ grid with a 50cm resolution. For training and validation, vehicles are considered only if their visibility level is above a predefined threshold (either 0% or 40%). To compare against other works, we refer the reader to Lift-splat [Phillion and Fidler, 2020] and CVT [Zhou and Krähenbühl, 2022].

Method	Conference	visibility > 0%	visibility > 40%	
		Setting 2	Setting 1	Setting 2
Lift-splat [Phillion and Fidler, 2020]	ECCV’20	32.1	—	—
CVT [Zhou and Krähenbühl, 2022]	CVPR’22	—	37.5	36.0
LaRa (ours)	CORL’22	35.4	41.4	38.9

cameras with a very general architecture that does not necessitate any strong geometric assumptions. Besides, when compared with CVT, we observe that LaRa obtains better results in the setting with finer resolution (+10% in Setting 1 vs. +8% in Setting 2).

Since our attention mechanism does not rely on all-to-all attention between camera images and BEV map as CVT does, LaRa can directly decode to the final BEV resolution which helps for fine prediction at a high resolution.

4.4.3 Extension to the driveable area segmentation task

In this section, we also provide results for the driveable area segmentation task, also addressed by CVT [Zhou and Krähenbühl, 2022]. Contrary to vehicle segmentation, this task requires the network to do “amodal completion” to a high degree, i.e., to correctly estimate regions of the road despite parts of it being severely occluded.

We followed the protocol of CVT [Zhou and Krähenbühl, 2022] for this segmentation task; the ground truth is generated using HD-map’s polygons from the dataset. We kept the same hyperparameters as used for the vehicle segmentation task, with a minor difference to the learning rate: we divide it by a factor 10 after 15 epochs (compared to a constant learning rate for vehicle segmentation).

TABLE 4.2: **Driveable area segmentation.** Results (in IoU) on nuScenes.

Method	IoU
CVT	74.3
LaRa (ours)	75.2

Quantitative and qualitative results for this additional task are given respectively in table 4.2 and fig. 4.2. When compared with CVT, we observe that LaRa achieves better performance (+0.9). Note that we do not do multi-tasking: following CVT [Zhou and Krähenbühl, 2022], we train a model specifically for the task of driveable area segmentation; the qualitative examples in fig. 4.2 are produced by fusing predictions from two models.

4.4.4 Model ablation and sensitivity to hyper-parameters

Input and Output-level embeddings. To assess the contribution of the geometric embeddings that we use, we compare the different choices at both the input and output level in table 4.3. As hypothesized, embedding the geometric relationship between cameras in the input is better suited for our task than the generic sine and cosine spatial embeddings. The additional camera index, while performing better than Fourier feature alone, is not



FIGURE 4.2: **Qualitative results on complex scenes.** We show the six camera views surrounding the vehicle along with segmentation ground truth for reference. Vehicles are shown in blue and driveable area in gray. Vehicles and driveable area predictions are from two different models trained independently for their respective ground-truth, the predictions are then merged for visualization purpose. The ego vehicle is located in the center and facing upwards. Predictions of both driveable area and vehicle segmentation are thresholded at 0.5 for visualization purpose.

enough to link pixels across cameras. For the output query embedding, the combination of normalized coordinates and radial distance gives the best results. This simple choice outperforms both the Fourier features [Jaegle et al., 2022; Vaswani et al., 2017] and learned embeddings [Carion et al., 2020; Vaswani et al., 2017] that also have the disadvantage of increasing the number of parameters.

TABLE 4.3: **Ablation study for the input and output query embedding.** Training and evaluation are done in Setting 2 (100m \times 100m at 50cm resolution), with a visibility $> 0\%$.

Input geometry embedding				Output query embedding				
Cam. rays	Cam. idx	Fourier	IoU	Radial dist.	Norm. coords	Fourier	Learned	IoU
✓	✗	✗	35.4	✓	✓	✗	✗	35.4
✓	✓	✓	34.4	✗	✓	✗	✗	35.1
✗	✓	✓	32.3	✗	✗	✓	✗	30.6
✗	✗	✓	30.5	✗	✗	✗	✓	21.8

Comparison to PETR embedding. In addition to our baselines, we include quantitative results in table 4.4 to compare our ray embedding against PETR [Liu et al., 2022a] embedding, a work concurrent to ours that also infuse “geometric” information into the “visual” stream to fuse information between views.

In PETR [Liu et al., 2022a], the embedding of each pixel is computed by sampling its ray given D predefined depths. The 3D coordinates of the D sampled points along the ray are normalized, concatenated, processed by an MLP and summed with the visual features. Conceptually, the embedding is a way to indicate to the network “this pixel can observe these 3D points in the camera frustum space”. The embedding in PETR differs to ours in that it is limited by the sampling resolution (i.e., the D predefined depths), as computation and memory footprint increase linearly with respect to D .

We trained our model with PETR input embedding in place of ours and show that our constant-complexity embedding is effective as a 3D positional embedding and performs better (+2%).

TABLE 4.4: **Impact of ray embedding on performance.** Vehicle segmentation performance (in IoU) for vehicle segmentation on nuScenes.

Embedding	IoU
PETR [Liu et al., 2022a]	34.8
Cam. rays (ours)	35.4

Sensitivity to hyper-parameters. To delve into the influence of hyper-parameters, we conduct a sensitivity analysis in fig. 4.3 where we vary the number N of latent vectors, their dimension M and the number of self-attention blocks L . We clearly observe that the performance increases with the number of latent vectors used. This is expected as it is the main parameter controlling the attentional bottleneck between input and output. Such a parametrization allows for an easy tuning of the performance/memory trade-off. We observe no clear correlation between the dimension M of latent vectors, the number L of self-attention layers, and the obtained IoU performance. This indicates that our architecture is not too sensitive to these hyper-parameters and can work efficiently with a wide range of values for these parameters. Although we obtain better results with 512 latent vectors, we use a maximum of 256 to stay in the same computational regime as the baseline we compare against; training with 512 latent vectors requires 32GB GPUs.

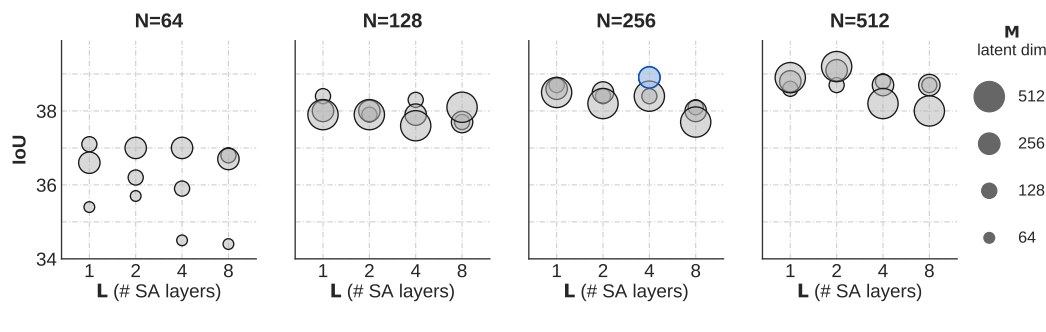


FIGURE 4.3: **Sensitivity study of LaRa to hyper-parameters.** We vary the number of latent vectors (N), their dimension (M), and the number of self-attention layers (L) and report IoU performances.

4.4.5 Study of attention

As quantitatively studied in section 4.4.4, embedding camera rays impacts significantly the performance of LaRa. In this section, we further support our claim that “our network is able to retrieve the pixel relationships between views thanks to our ray embedding” (Sec. 4.3) through a qualitative and quantitative analysis of the input-to-latent attention.

Attention qualitative analysis We further investigate the geometric reasoning of LaRa in [fig. 4.4](#) by analyzing the input-to-latent attention map. In this figure, we show two representations of the attention: a reprojection of the attention in the camera-space (left) and a top-view projection of the attention in polar coordinates by collapsing, i.e., averaging the vertical dimension (right). In the latter, the radial distance is proportional to the attention level and shows the directions the network attends the most.

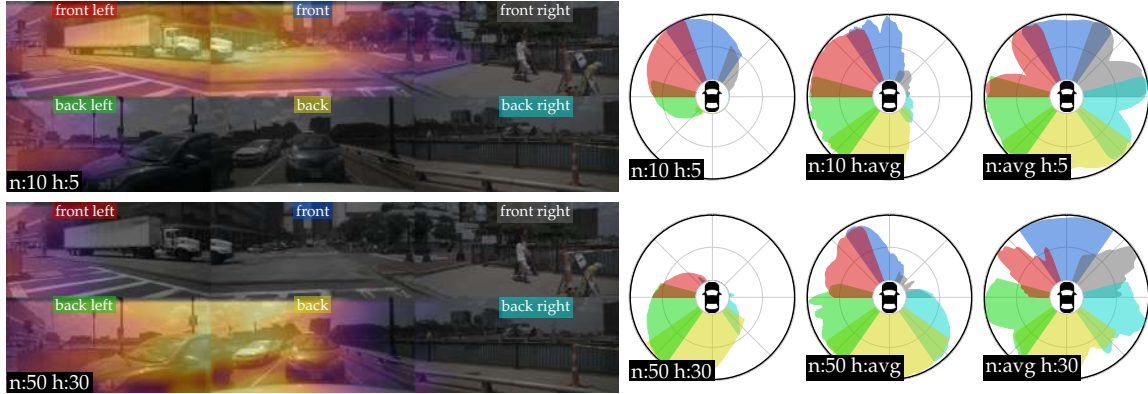


FIGURE 4.4: **Input-to-latent attention study.** Attention maps analysis for a network using 256 latents and 32 attention heads. The attention for one attention head and one latent is shown on the left superimposed with RGB images. The polar plots represent the directional attention intensity for one (or the average) attention head with one (or the average) latent vector. The radial distance is proportional to the attention level and shows the directions the network attends the most.

The study is conducted at three distinct levels. First, for a couple of one latent vector and one attention head ($n = 10, h = 5$ and $n = 50, h = 30$), among $N = 256$ possible latents and $H = 32$ possible attention heads. Second, for one latent vector and the averaged attention from all attention heads ($n = 10, h = \text{avg}$ and $n = 50, h = \text{avg}$). Third, for one attention head and the averaged attention over all latents ($n = \text{avg}, h = 5$ and $n = \text{avg}, h = 30$). From these three settings, we note the followings: First, the attention map between one latent vector and one attention head targets a specific direction (about a 90° field of view). Additionally, it can be clearly observed that the attention is continuous across cameras, proving the network is able to retrieve the pixel relationships between views. Second, while one attention head fires in a specific direction, the attention averaged over all the heads for one latent vector spans over half of the scene. This allows one latent vector to extract long-range context between views with the capacity to disambiguate them. Third, the attention for one head aggregated over all the latent vectors covers all directions, suggesting that the latent vectors contain all of the directional information and that the whole scene is attended across the latents. To summarize, by integrating early multi-view geometric cues instantiated by camera rays embedding ([section 4.3.1](#)), we show that LaRa learns to reason across views.

Additionally, we provide qualitative examples of the ‘Fourier + Cam. idx’ embedding to compare against our ray embedding in [fig. 4.5](#). Contrary to the attention yield by our ray embedding, the one derived from the ‘Fourier + Cam. idx’ embedding is much more spread out and less consistent across cameras.

Attention quantitative analysis. We now introduce a metric that directly quantifies the consistency and alignment of attention values across camera by analyzing behavior in “overlapping” regions, i.e., regions seen by two different cameras. We provide a visual description of this metric and its computation in [fig. 4.6](#).



FIGURE 4.5: **Input-to-latent attention study — influence of the input embedding.** Analysis of attention maps for two networks trained with different input embeddings. Top row is with ‘Fourier + Cam. idx’ and bottom row is with our proposed ‘Cam. rays’ embedding. The attention for one attention head and one latent is shown on the left superimposed with RGB images. The polar plots represent the directional attention intensity for one attention head with one latent vector. The radial distance is proportional to the attention level and shows the directions the network attends to the most.

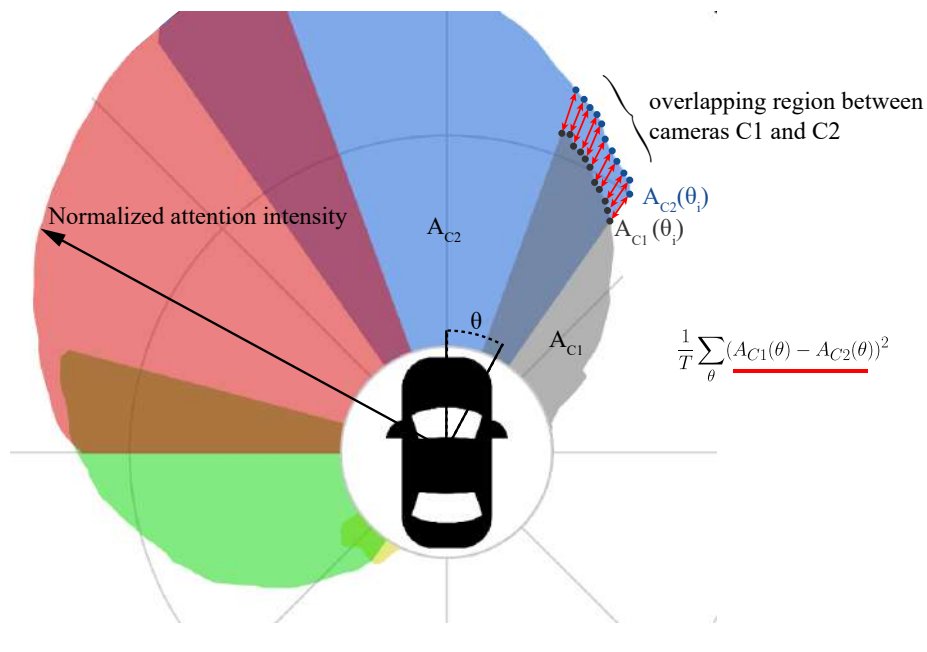


FIGURE 4.6: **Measuring the attention consistency across cameras.** The proposed metric computes the Mean-Squared-Error (MSE) of the attention intensity on overlapping regions between cameras (as illustrated for two cameras and one latent and one attention head), and averages it over all cameras, latents, heads and scenes.

In short, knowing the orientation of each camera, we compute the Mean Squared Error (MSE) of the directional attention intensity between cameras on their overlapping regions.

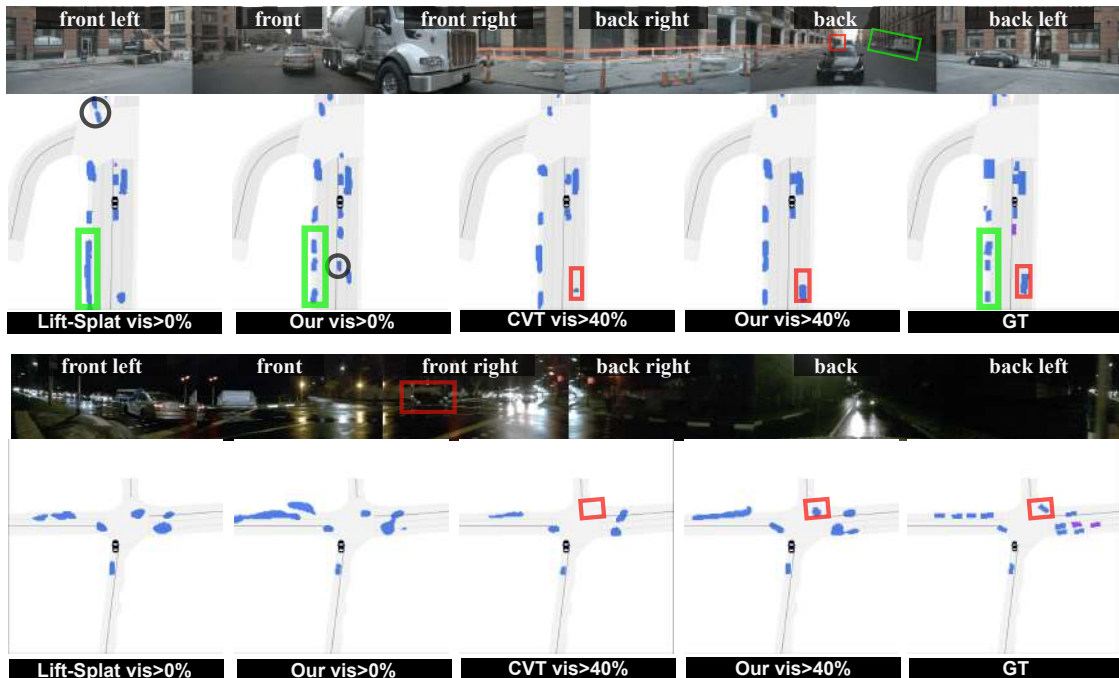


FIGURE 4.7: **Qualitative results on complex scenes.** We show the six camera views surrounding the vehicle along with segmentation map ground-truth for reference. In the ground-truth (GT) map, vehicles are shown in blue (visibility $> 40\%$) or purple (visibility $< 40\%$). The ego vehicle is located in the center and facing upwards. We show results for our two baselines [Phillion and Fidler, 2020; Zhou and Krähenbühl, 2022]. For a fair comparison, we always compare using their respective level of visibility. Setting 2 is used.

This score is averaged for all the overlapping regions, latents and attention heads, and examples in the validation set. A score of zero indicates a perfect match of the attention levels on overlapping regions (i.e., across cameras). Results with this metric, reported in table 4.5, show that our ‘Cam. rays’ embedding is 10 times more “consistent” across cameras than the baseline ‘Fourier + Cam. idx’.

TABLE 4.5: **Impact of ray embedding on cross-camera attention consistency.** Cross-camera attention consistency (measured with proposed MSE metric, see Fig. 4.5) on nuScene.

Embedding	MSE on overlap
2D Fourier + Cam. idx	0.0896
Cam. rays (ours)	0.0068

4.4.6 Qualitative Results

We show the segmentation results of two complex scenes in fig. 4.7. For a fair comparison, we use our model trained with visibility $> 40\%$ against CVT and $> 0\%$ against Lift-Splat. Compared to LaRa, CVT missed two objects, one at a long distance and the other in the dark (red box). We also estimate the boundaries of the vehicles better than Lift-Splat (green box). Interestingly, models trained on all vehicles (visibility $> 0\%$) tend to hallucinate cars in occluded or distant regions (highlighted with black circles in the figure).

4.5 Extension to temporal modelling

So far, LaRa demonstrates that the geometry and semantics of a complex scene can be compacted in a small collection of latent vectors. We believe that this formulation would allow for efficient temporal reasoning. By retaining information from the past, the system can better refine the information and improve overall performance. In particular, it may help to handle challenging situations like when a car is currently occluded but has been seen in past frames, to refine the estimated boundaries objects (e.g., in [fig. 4.7](#) some vehicles are elongated), and to better detect small and distant objects.

4.5.1 Additional modules

Current methods [[Hu et al., 2021](#); [Li et al., 2022b](#)] carry out the temporal modelling in the BEV space, which is of high resolution, costly to process, while mostly representing empty space. As the ego-car moves, there is a need to align each past BEV feature maps into the current ego reference frame to take in account the change of point of view. Typically, this alignment (or ‘warping’) is operated using ego-motion information, by a simple rotation and translation of the BEV feature maps.

In contrast, we take advantage of our implicit representation and efficiently aggregate information over time directly in the latent representation. However, the latent representation being an abstract space, we cannot “warp” the information of the scene. Hence, we extend our LaRa architecture with two additional modules: (1) a module to integrate the ego-motion information, and (2) a recurrent update of the latent representation (see [fig. 4.8](#)). The ‘motion conditioning’ module takes as input the ego-motion of the car to make the internal representation “motion-aware”. It is necessary to integrate the ego-motion for the network to extract dependencies between viewpoints across time. Otherwise, not able to make sense of it, the network simply ignores information from the past. The ‘recurrent update’ takes information from the past and integrates it with the information currently observed. This allows refining the current prediction and better handling occlusions.

More formally, we reduce the ego-motion to a horizontal motion: a rotation of angle θ with respect to the vertical axis and a translation in the horizontal plane $\Delta x, \Delta y$. These parameters, defining the relative change in pose between two time steps $P_{i-1 \rightarrow i} = [\cos(\theta), \sin(\theta), \Delta x, \Delta y]$, are given as input to a 2-layer MLP which lifts $P_{i-1 \rightarrow i}$ from 4 dimensions to M dimensions, i.e., the size of the latent vectors. This MLP produces an “ego-motion feature vector” that is then infused in the latent representation using a cross-attention. More specifically, the embedding of $P_{i-1 \rightarrow i}$ is given as the key and value to the cross-attention and the past latent representation as the query. This, now “motion-aware” latent representation, feeds a recurrent module to aggregate information over time. In our experiment, we instantiate the recurrent module as a simple Gated Recurrent Unit (GRU [Cho et al. \[2014\]](#)).

4.5.2 Results

In [table 4.6](#) we report results for vehicle segmentation and inference time against FIERY [[Hu et al., 2021](#)]. FIERY builds on Lift-splat [[Phillion and Fidler, 2020](#)] (a baseline in [table 4.1](#)) and extends it for temporal modelling. FIERY aggregates information over time in the BEV space. For this, it first computes BEV feature maps using the method of Lift-splat [[Phillion and Fidler, 2020](#)]. Then it uses ego-motion information to warp (i.e., rotating and translating) each past BEV feature maps into the current ego reference frame. This creates a spatio-temporal volume of T BEV feature maps, which is then processed by a 3D convolutional layer. The 3D convolutions extract spatio-temporal patterns from the BEV feature

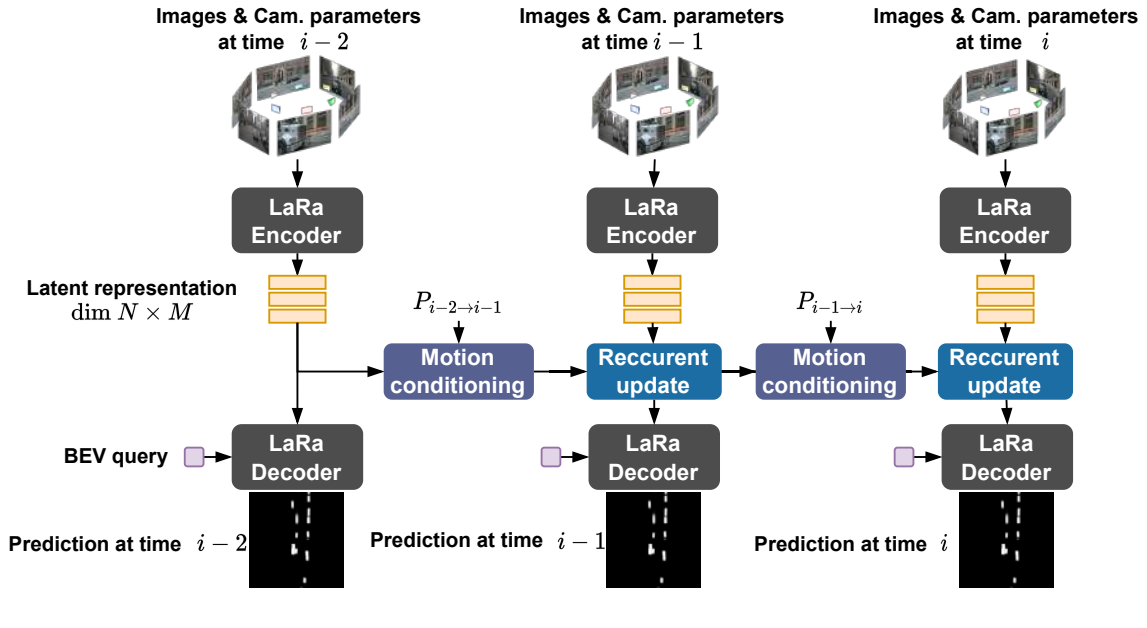


FIGURE 4.8: **LaRa temporal**. Schematic illustration of the temporal extension of LaRa. The ‘motion conditioning’ module takes as input the ego-motion of the car to make the internal representation “motion-aware”. The ‘recurrent update’ takes information from the past and integrates it with the information currently observed.

TABLE 4.6: **Results of temporal integration**. We present results for vehicle segmentation performance (in IoU) for vehicle segmentation on nuScenes and the inference time in milliseconds for a forward pass on a V100 GPU. Note that FIERY [Hu et al., 2021] initially address the task of instance segmentation; here we re-train the model for binary segmentation of vehicles, all other parameters remain unchanged. In particular, the models take the past two frames in addition to the current frame for temporal modelling.

Method	IoU	Forward pass (ms)
FIERY [Hu et al., 2021]	37.5	550
LaRa temporal (ours)	37.1	70

maps, allowing to reason about the dynamic of the vehicles in the scene. This produces a single spatio-temporal BEV feature map (i.e., the temporal dimension is collapsed) that is then used to make the final prediction. This approach requires processing the T past BEV feature maps in parallel.

In contrast, our LaRa temporal model processes the scene information sequentially in the latent space. With this approach, we obtain competitive IoU performances while having an inference $8\times$ faster.

4.6 Conclusion

In this chapter, we proposed LaRa, a general architecture for scene understanding from multiple cameras.

4.6.1 Summary of contributions

Sensor fusion and capacity to efficiently represent a scene. This work demonstrates that *some* of the geometric and semantic information of a complex scene, can be efficiently aggregated from many sensors and encoded in a very compact, but rich, implicit representation.

Ray embedding. By incorporating ray embeddings into LaRa, we augment visual features with geometric cues of the scene and show that this leads to multi-view stitching (supported by our attention analysis in [section 4.4.5](#)) and improved performances over other baselines.

Robustness to hyper-parameters. Our architecture has been tested with a broad range of hyper-parameters ([section 4.4.4](#)) and has also been successfully implemented on a prototype car, which has a different number and arrangement of cameras. These experiments demonstrate the adaptability of LaRa and its capacity to be deployed in various conditions.

4.6.2 Perspectives

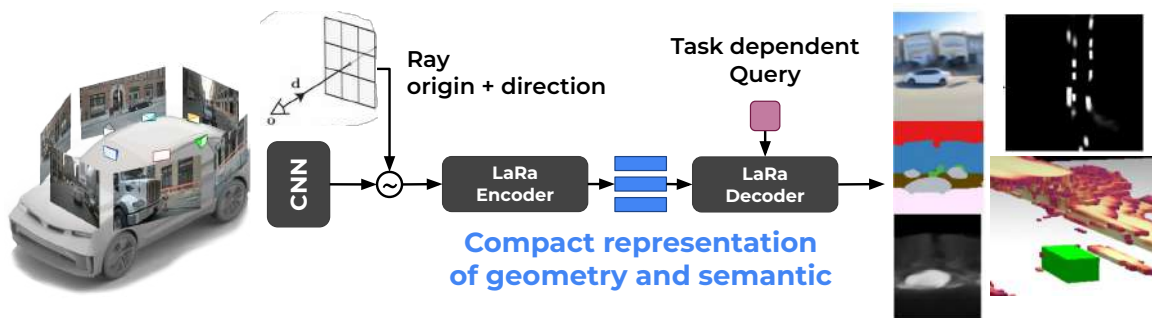


FIGURE 4.9: **LaRa meta-architecture.** Schematic illustration of how our architecture could be applied to a wide range of tasks. Here we depict outputs for view reconstruction, pixel-wise segmentation, depth estimation, BEV vehicle segmentation, and 3D occupancy.

Extension to other tasks. LaRa is not limited to BEV predictions. In particular, using an appropriate query, predictions could be done in any other output space, including depth or semantic prediction in the image plane or 3D occupancy grid prediction ([fig. 4.9](#)).

Evaluation against visual impediments. We demonstrated the very good capacity of our architecture to extract and fuse complex and long-range dependencies (even across cameras). Our method could further be evaluated against depth-based methods in difficult visual conditions (e.g., water droplets or soiling on the camera lens, glares like in [fig. 2.7](#), etc.).

Sensor Fusion. Our model operates on camera inputs only, a setup in which correctly estimating distances is difficult ([chapter 3](#)). Moreover, in adverse conditions, e.g., with glares and darkness, its performance may remain limited. To better handle these challenging situations, one avenue of improvement would be the extension of LaRa to handle complementary modalities, e.g., coming from LiDARs or radars.

Chapter 5

Conclusions and future directions

5.1 Conclusions and discussions

In this PhD thesis, we explore the use of deep learning for scene understanding applied to autonomous driving. In particular, we address several important research questions (RQs) including:

- RQ1.** How to leverage inexpensive sensors (e.g., camera, minimal 4-beam LiDAR, etc.)?
- RQ2.** How to fuse information from multiple sensors?
- RQ3.** How to alleviate the need for annotated data?
- RQ4.** How to estimate a map of the environment in real time from raw sensors?

First, in [chapter 3](#), we tackle the important task of depth estimation. To take full advantage of the abundant flow of unannotated data that a fleet of cars can produce, our method is self-supervised (**RQ3**). Leveraging inexpensive sensors typical of existing passenger cars, we show competitive performances compared to methods relying on expensive sensor suites and trained in a fully-supervised way (**RQ1**). Given a camera and a minimal LiDAR with as few as 4 beams (**RQ2**), our system can learn to estimate depth on any domain without any annotation.

Our work in [chapter 3](#) also highlights the difficulty to predict accurate pixel-wise depth, a very *explicit* way to represent the geometry of the scene. This pushed us to explore a more *implicit* approach. The main objective of [chapter 4](#) is to create an architecture able to encode the geometric and semantic information of a complex scene in a very compact, but rich, internal representation. This effort led to ‘LaRa’, a transformer-based architecture for scene understanding. Thanks to its ray embedding, it is able to encode information from many sensors into a small latent representation of the scene (**RQ2**). We can then process and query information from this compact representation to efficiently segment vehicles and driveable areas in the Bird’s-Eye-View space (**RQ4**). Moreover, in [section 4.5](#) we provide evidence that knowledge about the scene can be accumulated over time directly in this abstract representation of the world; paving the way for models that efficiently reason and plan in the latent space.

Despite our work making progress on the topic of scene understanding for autonomous driving, there are still several avenues for future work. We comment on possible extensions and outline future research directions in the following.

5.2 Future directions

5.2.1 Handling multiple types of cameras and different intrinsics.

Across datasets [[Caesar et al., 2020](#); [Geiger et al., 2012](#); [Sun et al., 2020](#); [Yogamani et al., 2019](#)], images from cameras come in all different kinds of shapes and sizes (e.g., pinhole,

fish-eye, etc.). When the intrinsics of the camera change, the way the light interacts with the camera sensor is also modified. This affects the way visual data is captured, which ties the filters learned by the CNN to the intrinsics present in the learning data. A naïve, time-consuming, and computationally expensive way to address this issue is to retrain the entire network from scratch or to fine-tune it with the new data.

To build a more robust and efficient autonomous driving system, the works ‘CAM-ConvS’ by [Facil et al. \[2019\]](#) and ‘Camera Tensor’ by [Ravi Kumar et al. \[2021\]](#) offer a way to handle these different types of cameras without having to retrain the whole model for each. The CAM-ConvS method proposes to inject the intrinsic parameters of the camera in the CNN as additional channels of the feature maps (e.g., the distance to the camera principal point is concatenated to each pixel). By doing so, the network can learn to account for visual variations due to changes in the intrinsics of different cameras. Camera Tensors [[Ravi Kumar et al., 2021](#)] builds upon this work and generalizes to arbitrary camera geometries, including fisheye cameras

Combined with our ray embedding ([chapter 4](#)), which already integrates the intrinsics and extrinsics of the cameras, such approaches would further increase the versatility and adaptability of LaRa, allowing its deployment on a wide range of mobile robots.

5.2.2 Leveraging Simulation

While the core of this thesis is not about planning and control for autonomous driving, it is essential to keep in mind that our end goal is not to achieve high performance on individual benchmarks, but to ensure safe and efficient motion of the vehicle. Indeed, a higher performance in benchmarks for multi-modal motion forecasts may not translate into better and safer motion planning [[Casas et al., 2020](#)]. Hence, we need ways to evaluate our complete system in realistic driving conditions. Likewise, it is important to evaluate the performance of our methods in a closed-loop setup as open-loop evaluations, while proposed in certain benchmarks [[Caesar et al., 2021](#)], do not take into account the interactions between the vehicle and the environment. Since we cannot train and evaluate closed-loop systems on public roads for security reasons, validation on test tracks and in simulation is necessary. Note that the simulator does not necessarily need to be photorealistic, but it needs to capture and represent the mutable characteristics of the world [[Sun et al., 2022](#)] (e.g., weather, time of day, agent density, road layout and agent behaviour). In this regard, CARLA [[Dosovitskiy et al., 2017](#)] is a simulator that generates diverse virtual environments closely resembling real-world scenarios, including realistic weather conditions, traffic, and road layouts. It also simulates a wide range of sensors including cameras, LIDARs, and radars, which can be used to evaluate the performance of different sensor configurations available on real vehicles [[Hu et al., 2022b](#)]. These elements motivate the development of better evaluation standards, and publicly available simulators such as CARLA [[Dosovitskiy et al., 2017](#)] appear to be promising tools to develop and test autonomous driving algorithms.

5.2.3 Learning an implicit representation of the world

In [section 2.3.4](#) we introduced the notion of an implicit representation of the world. The supposed advantages of such a representation are numerous, including the removal of irrelevant details; facilitating the extraction of complex dependencies. It alleviates any resolution issue by projecting in a continuous and abstract space. It also extracts the statistical regularities of the world, something that helps model the temporal dynamic of the world and reduces the amount of annotated data necessary to train on a particular task [[Hafner](#)

et al., 2023]. To achieve this goal, two main problems must be solved. First, we need a neural network architecture that can compress raw sensor signals into a compact encoding. Second, we need a system to train this architecture so that this encoding has the properties stated previously.

For the architecture, LaRa, presented in [chapter 4](#), constitutes a first attempt to project multiple sensory streams into a small collection of latent vectors. This internal representation can then be queried to make predictions in an arbitrary output space; allowing a single representation to learn from multiple and diverse supervisory signals.

That leaves the second major question: how to learn such a representation? Learning such abstract encoding of the world naturally involves the compression of relevant information. However, the notion of “relevant information” only makes sense in regard to a task, i.e., the latent space and what information are discarded from the inputs are entirely defined by the supervisory signal. In this aspect, an implicit representation is also very flexible and supervision of many forms can be used. For example, in a predictive fashion, one may want to predict the future observation (e.g., at the next time step) directly at the latent state level for ease of environment modelling: a consistency between the future prediction and the actual observation is imposed on the latent vectors [[Sobal et al., 2022](#); [Ye et al., 2021](#)]. However, in dynamic and noisy environments, such an objective may be minimized by focusing on irrelevant correlations [[Sobal et al., 2022](#)]: the network may only encode parts of the scene easily predictable that change very slowly over time, like the sky, instead of learning the agents’ behaviours.

To address this drawback, a reconstruction objective is often used as an additional objective, the aim is to force the latent state to maximize the information it contains. For example, one of the most known approaches is reconstructing the input signal, or *auto-encoding*. However, reconstructing the entire RGB signal from cameras can be a very hard task due to visual variations and, above all, not necessarily useful (e.g., encoding the shape of the clouds or the texture of buildings does not help to drive). Instead, representations typically used in perception tasks can serve as additional supervisory signals. In particular, depth, optical flow, and semantic segmentation can be used to supervise the scene representation [[Zhou et al., 2019](#)]. These explicit representations provide a level of abstraction with far fewer details to predict (e.g., they do not present variations of albedo or lighting). This is helpful to infuse important concepts like 3D geometry and motion in the implicit representation. In particular, the work of [Zhou et al. \[2019\]](#) suggests it improves transferability to control tasks. At the extreme, the work of [Hu et al. \[2022a\]](#) supervises the internal representation with a BEV semantic occupancy grid encoding the rules of the road (traffic lights state, stop sign) and dynamic agents (vehicles and pedestrians). These approaches have the drawback of requiring a ground truth; learning these concepts in a self-supervised way remains an open problem.

In this regard, we showed that depth can be self-supervised in [chapter 3](#). Likewise, optical flow [[Yang and Ramanan, 2020](#)] and semantic segmentation [[Vobecky et al., 2022](#)] can also be self-supervised. More recently, [Wimbauer et al. \[2023\]](#) presented a self-supervised method based on view reconstruction and neural rendering to estimate the volumetric occupancy of a scene (i.e., estimating the 3D geometry of a scene even in occluded areas). All these approaches, requiring no annotation, could be exploited to learn a compact representation of the geometry, dynamics, and semantics of the world in a self-supervised way.

Appendix A

Monocular metric depth estimation with a few-beam LiDAR

A.1 Overfitting to input LiDAR

In this section, we provide qualitative examples as well as elements of analysis for the convergence behavior observed on ACMNet, NLSPN and S2D that we call “*overfitted to LiDAR input*”. To this end, we compare S2D (*overfitted*) to Monodepth2-L (*metric*) trained with a pose network and (‘P+L₄’) supervision for 30 epochs. In essence, we refer to models as *overfitted* when most of the depth prediction is consistent but only *relative*, while depth prediction is only metric on pixels with LiDAR data. On [fig. A.1b](#), we can clearly observe the difference in scale between areas with and without LiDAR data. Likewise, we can quantitatively observe the existence of two distinct scales within predictions of S2D. In the middle plot of [fig. A.2](#), the median value of the inverse depth prediction (disparity) on pixels with LiDAR are roughly the same for S2D and Monodepth2-L, they are both scaled metrically. On the other hand, in the top plot of [fig. A.2](#) showing the median value of disparity on pixels without LiDAR, there is a clear difference between Monodepth2-L, that is properly scaled, and S2D that converged to a random scale.

From a supervisory perspective, the depth network is stuck within a local minima where the photometric loss is mostly minimized apart on pixels with LiDAR data where it is clear the pixels are projected at different scale (see [fig. A.1a](#)). The amount of pixels with LiDAR data being very small, the erroneous photometric loss is on these areas is strongly dampened by the average over the whole image. So strongly dampened that that photometric loss between S2D and Monodepth2-L, respectively an *overfitted* and a *metric* model, almost perfectly match (see photometric loss [fig. A.3](#)). At the same time the LiDAR loss has already reached a minimum and the smoothness loss is not powerful enough to regularize this convergence behavior.

This convergence profile is expected because there are an infinite number of depth prediction scales for which the photometric loss is minimized over areas with no LiDAR data. Hence, there is an infinite number of local minima leading to this *overfitted* behavior. On the contrary, when using LiDAR self-supervision, only one depth prediction scale exists, the *metric* one, to obtain a globally coherent reconstruction. We propose a solution to this problem for S2D as well as ACMNet and NLSPN in [appendix A.3](#).

A.2 Ablation of LiDAR: further analysis

In this section, we analyse results for learning setups not described in [section 3.5.1](#). In particular, we continue to study the use of a pose network instead of PnP, with ‘P’, ‘P+IMU’ or ‘P+L₄+IMU’ supervision.

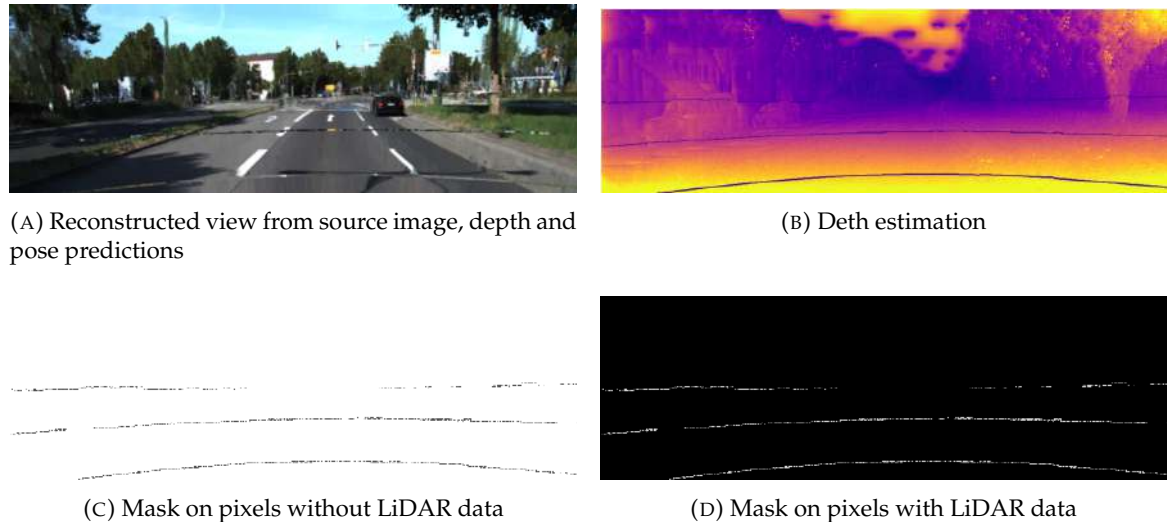


FIGURE A.1: **Predictions of a model overfitting to LiDAR input.** We show (A) a reconstructed view from source image, depth and pose prediction (B) a depth estimation considered as *overfitted* to the LiDAR input (C) a binary mask where the value is 1 for pixels without LiDAR data and 0 otherwise (D) a binary mask where the value is 1 for pixels with LiDAR data and 0 otherwise.

Overall, we observe very poor performances with the use of the pose network. First, we note that the use of photometric reconstruction only (‘P’ in table 3.2) leads to to **relative** depth for all networks (dark gray cells in the table 3.2). Indeed, this setup is well known as being an ill-posed problem [Godard et al., 2019; Guizilini et al., 2020a; Wang et al., 2018; Zhou et al., 2017]; the pose provided by the monocular pose network can only be relative without additional information, and the depth estimation is thus unscaled as well.

To enforce a metric scale, we train the pose network with additional supervision in the form of an IMU prior (‘P+IMU’), as explained in section 3.4.2. While this helps Monodepth2 and Monodepth2-L to correctly train, ACMNet, NLSPN and S2D architectures cannot reach good performances when a joint alignment between a pose and depth network is required (see appendix A.5 for more details).

With further supervision from the input LiDAR (‘P+L₄+IMU’), we can slightly increase results for Monodepth2 and Monodepth2-L as well as significantly boosting results for ACMNet compared to the (‘P+IMU’) setup (253% increase). However, similar to ACMNet, NLSPN and S2D in the (‘P+L₄’) setup (see section 3.5.1), NLSPN and S2D tends to overfit the input LiDAR. Hence, we use the same *dilation* procedure, as detailed in appendix A.3, for these models to avoid overfitting the LiDAR input.

A.3 Dilated LiDAR

Contrarily to Monodepth2 and Monodepth2-L, when trained with a pose network and LiDAR self-supervision, the networks ACMNet, NLSPN and S2D tend to overfit the LiDAR. Most of the depth prediction is consistent but only relative, while depth prediction on pixels with LiDAR data is metric (see appendix A.1 for an example). The main difference between these architectures is that Monodepth2 and Monodepth2-L are supervised at multiple scales (1:1, 1:2, 1:4 and 1:8) while ACMNet, NLSPN and S2D are only supervised at the final resolution (1:1). Supervision at the lowest scale (1:8) artificially increases the number of pixels getting supervision from LiDAR as a LiDAR point spans multiple pixels when projected at low resolutions.

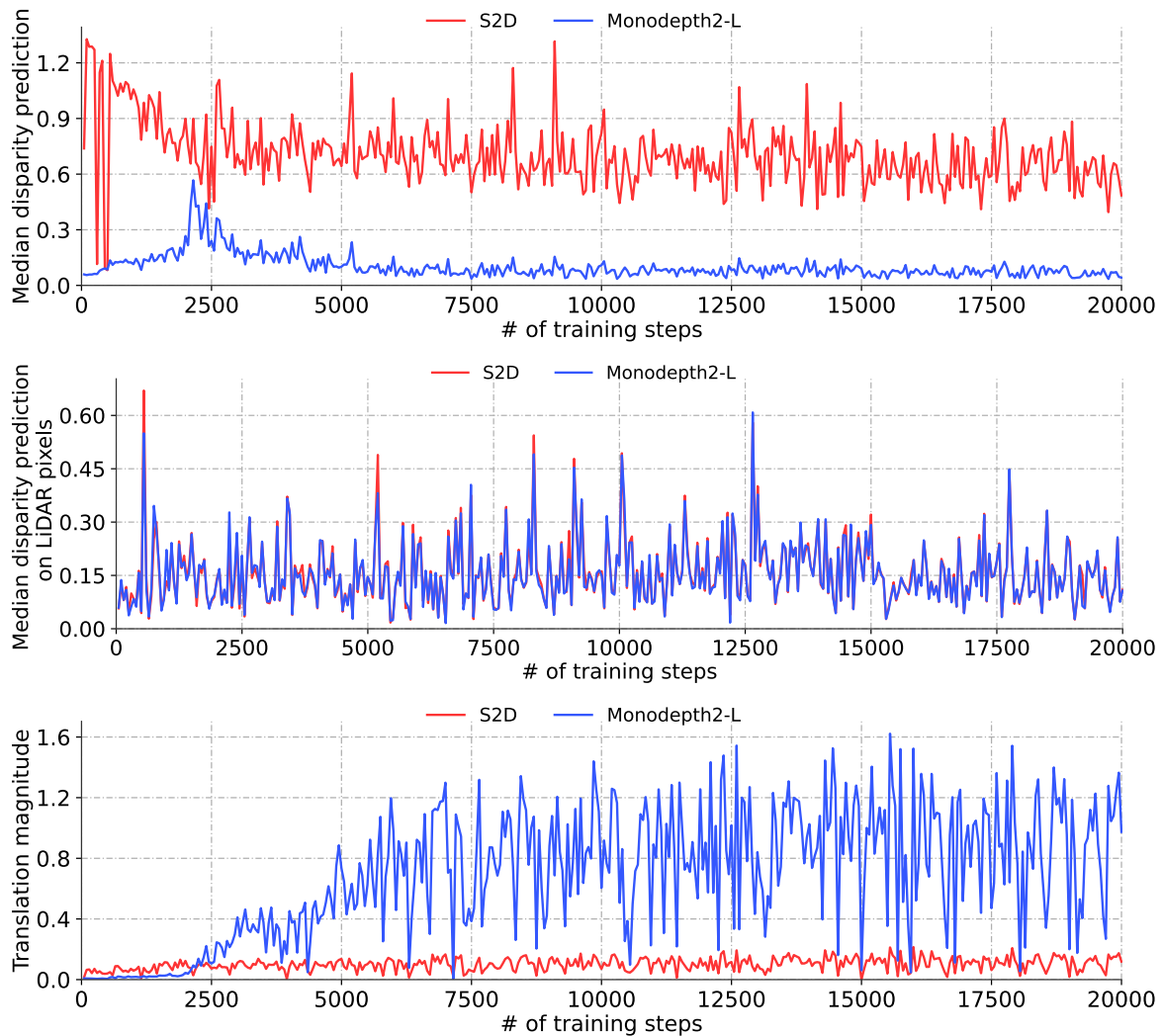


FIGURE A.2: Statistics for the depth and pose outputs over a training run for S2D (*overfitted*) and Monodepth2-L (*metric*) with a pose network and ('P+L_4) supervision. Respectively from top to bottom, we provide the median value of the disparity predicted by the depth network on pixels without LiDAR data (see Figure A.1c), then on pixels with LiDAR data (see Figure A.1d), and the magnitude of the relative pose's translation component (i.e., by how much the pose network estimates the car moved between two views).

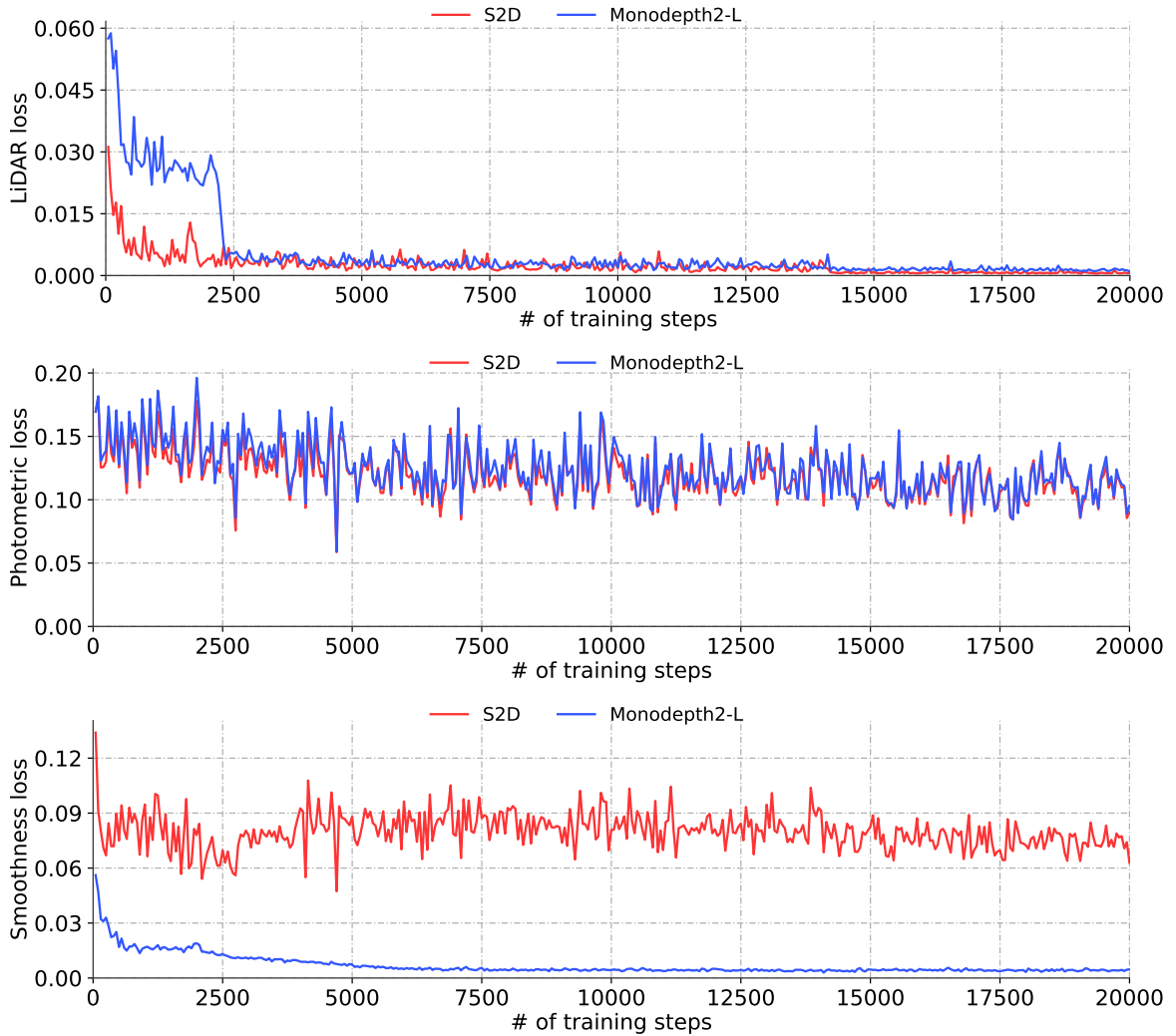


FIGURE A.3: Loss values over a training run for S2D (*overfitted*) and Monodepth2-L (*metric*) with a pose network and ('P+L₄') supervision. Respectively from top to bottom, we provide the values over a training run for the self-supervised LiDAR loss, the photometric loss as well as the smoothness loss.

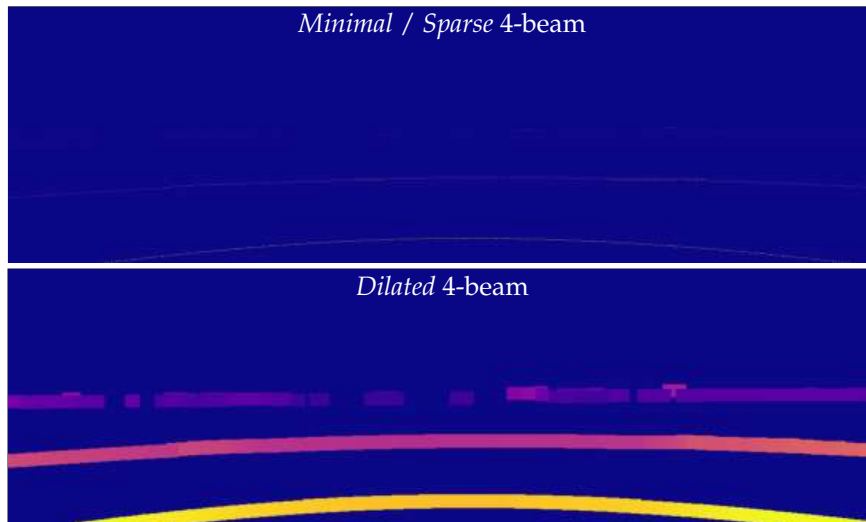


FIGURE A.4: Visual difference between vanilla and dilated LiDAR.

We hypothesize that the mono-scale training is the cause of overfitting to LiDAR input when training with LiDAR self-supervision. This is confirmed by the fact that, when Monodepth2-L is only supervised at the scale 1:1, the model collapses into the *overfitted* regime which highlights the importance of multi-scale training.

As modifying the mono-scale networks is non-trivial, we propose to self-supervise with a dilated LiDAR to compensate for the lack of multi-scale supervision and to avoid overfitting the LiDAR input. More precisely, we apply two iterations of a *dilation* morphological operator with a kernel of 10×10 on the 4-beam LiDAR at the supervision level only (i.o.w., we do not apply *dilation* on the LiDAR input). The aim is to increase the number of pixels receiving LiDAR supervision, albeit in a noisy manner, (fig. A.4). This simple procedure, while remaining a trick, enables mono-scale architectures to avoid overfitting the input LiDAR and to converge to metric depth estimation. On the other hand, none of the architectures need such special care when trained under our LiDARTouch framework. We report results of models trained with this procedure with the superscript † in table 3.2.

In addition to this strategy, we explored various experimental setups and combination of hyper-parameters when training with (P+L4) and (P+L4+IMU) for mono-scale networks:

- Dividing the sparse LiDAR depth values (used as input and/or ground-truth) by a factor α at train time and multiply depth prediction consequently at validation. The network still overfits to LiDAR data with $\alpha \in \{10, 100, 1000\}$.
- Decreasing the contribution of the depth loss in the global objective to mitigate the overfitting behavior to LiDAR points. With $\lambda \in \{1, 1e-1, 1e-2, 1e-3\}$, the model still overfits the LiDAR. With $\lambda \in \{1e-4, 1e-5\}$ the network stops overfitting the LiDAR data but the depth estimation becomes only relative instead of being metric.
- Increasing the contribution of the smoothness loss in the global objective. By doing so, we hoped to uniformize the scale of the depth prediction on pixels without LiDAR that are neighbors to pixels with LiDAR. The network still overfits to LiDAR data with $\lambda \in \{1e-1, 1e-2, 1e-3\}$.
- Varying learning rate from $1e-3$ to $1e-5$. The network still overfits to LiDAR data.

A.4 Pose scaling is critical when using a PnP pose estimation with photometric loss only

Most of the depth network’s learning signal comes from the reconstruction of the target image from the source image. For a given scale, a correct photometric reconstruction corresponds to a unique pair of depth and pose. Hence, for one to be metrically scaled, both the depth and the pose have to be metric. However, the networks are initialized randomly and thus need to jointly align and converge to a metric scale.

On the other hand, when using PnP, the estimated pose is metric thanks to LiDAR data (see [section 3.3.3](#)), thus, only the depth network has to converge to the correct scale. However, this may produce a large difference in scale at initialization between the pose and depth, provoking unstable training for the depth network. Thus, one strategy we adopt to stabilize training is to divide the translation component of the PnP pose by 10 and multiply the depth prediction by 10 at inference time. Models trained with this strategy are indicated with the superscript * in [table 3.2](#).

To circumvent these difficult training behaviors, we can use the PnP method to produce metric poses, and further enforce the collapse of the depth solutions to a metric scale with additional LiDAR self-supervision. This is consistently verified with the use of photometric and LiDAR supervisions (P+L₄) for each of the five architectures considered and leads to the best results compared to any other configuration (see [table 3.2](#)). These results demonstrate that the use of LiDAR both as self-supervision and in pose computation yields performance on-par or better than camera-only setups.

A.5 Poor performances for ACMNet, NLSPN and S2D when trained with P+IMU

Unfortunately, we cannot make these models converge to metric depth estimations. We describe below the combination of hyper-parameters we experimented with:

- Dividing the pose GT (translation magnitude) by 10, 100, 1000 and multiplying depth predictions consequently.
- Varying the contribution of the smoothness loss with $\lambda \in \{1e-1, 1e-2, 1e-3\}$.
- Varying learning rate from $1e-3$ to $1e-5$.

In all these cases, **the networks still converge to bad quality depth estimations**. We also investigate Monodepth2-L only supervised at the biggest scale to evaluate the influence of multi-scale training in the ‘P+IMU’ setup. We found that performances slightly decreased, but the network still converges to metric depth estimations. Hence, in this setup, multi-scale training does not seem to be crucial.

Appendix B

Latents and Rays for an Implicit Scene Representation

B.1 Output embedding

In [chapter 4](#), we considered Fourier features and learned query as alternative BEV query embeddings. Here we detail both of them.

Fourier features. The Fourier encoding has been proven to be well suited for encoding fine positional features [[Jaegle et al., 2022](#); [Vaswani et al., 2017](#); [Yifan et al., 2022](#)]. This is done by applying the following on an arbitrary input $z \in \mathbb{R}$:

$$\text{fourier}(z) = (z, \sin(f_1\pi z), \cos(f_1\pi z), \dots, \sin(f_B\pi z), \cos(f_B\pi z)), \quad (\text{B.1})$$

where B is the number of Fourier bands, and f_b is spaced linearly from 1 to a maximum frequency f_B and typically set to the input’s Nyquist frequency [[Jaegle et al., 2022](#)]. The maximum frequency f_B and number of bands B are hyper-parameters. This Fourier embedding is applied on the normalized coordinate grid such that:

$$Q_{\text{fourier}}[i, j] = \text{fourier}(Q_{\text{coords}}[i, j]_i) \oplus \text{fourier}(Q_{\text{coords}}[i, j]_j). \quad (\text{B.2})$$

Learned. Another alternative, following common transformer practice [[Carion et al., 2020](#); [Vaswani et al., 2017](#)] and most notably proposed by CVT [[Zhou and Krähenbühl, 2022](#)], is to let the network learn its query of dimension $d_{\text{bev-query}}$ from data. However, this is memory intensive as it introduces $h_{\text{bev}} \times w_{\text{bev}} \times d_{\text{bev-query}}$ additional parameters to be optimized. In other words, the number of parameters grows quadratically to the resolution of the BEV map. For experiments using learned output query embedding, we use $d_{\text{bev-query}} = 32$.

B.2 Additional attention qualitative analysis

We also provide additional analysis of attention maps for the multi-camera input shown in [fig. B.1](#) with a network using 256 latents and 32 attention heads. As in [chapter 4](#), the polar plots represent the directional attention intensity, showing the directions the network attends the most. The contribution of each camera is indicated by a color code coherent with [fig. B.1](#). Each polar plot is oriented in an upward direction (i.e., the front of the car points upward).



FIGURE B.1: Six input camera images coming from the 360-degree camera rig of nuScenes. Note the small overlaps between views, e.g., the front of the white truck is both seen in the front-left and front cams.

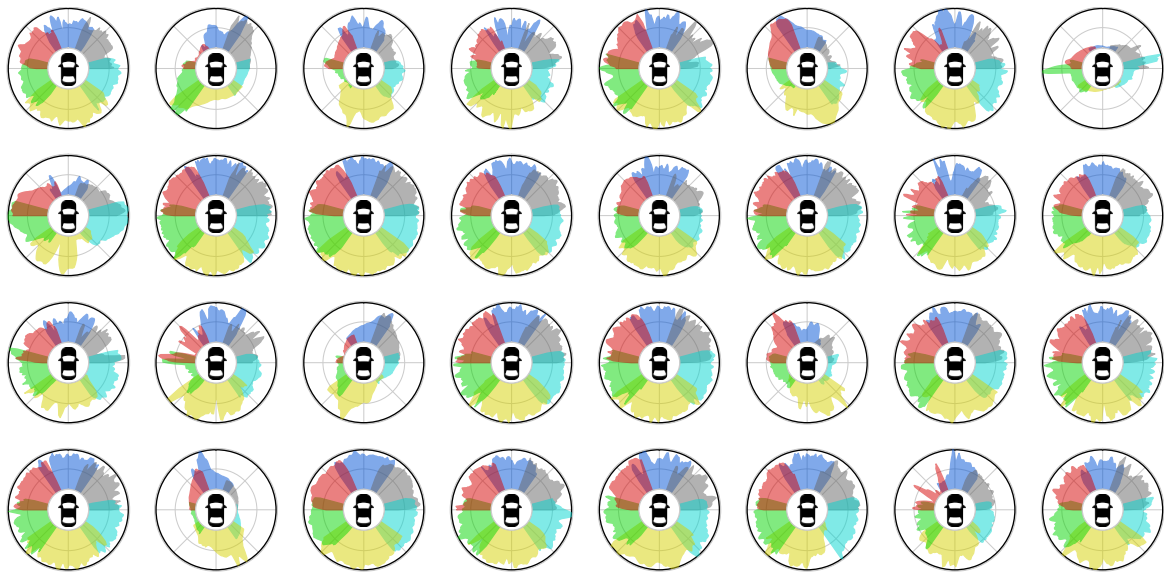


FIGURE B.2: **Input-to-latent attention study — average over latents.** These polar plots represent the directional attention intensity averaged over all the 256 latent vectors for each attention head. When averaging over latent vectors, we observe that each attention head generally covers all directions. This suggests that the latent vectors contain most of the directional information and that the whole scene is attended across the latent. More rarely, an attention head’s polar plot will be directional but will maintain a level of generality by being symmetrical.

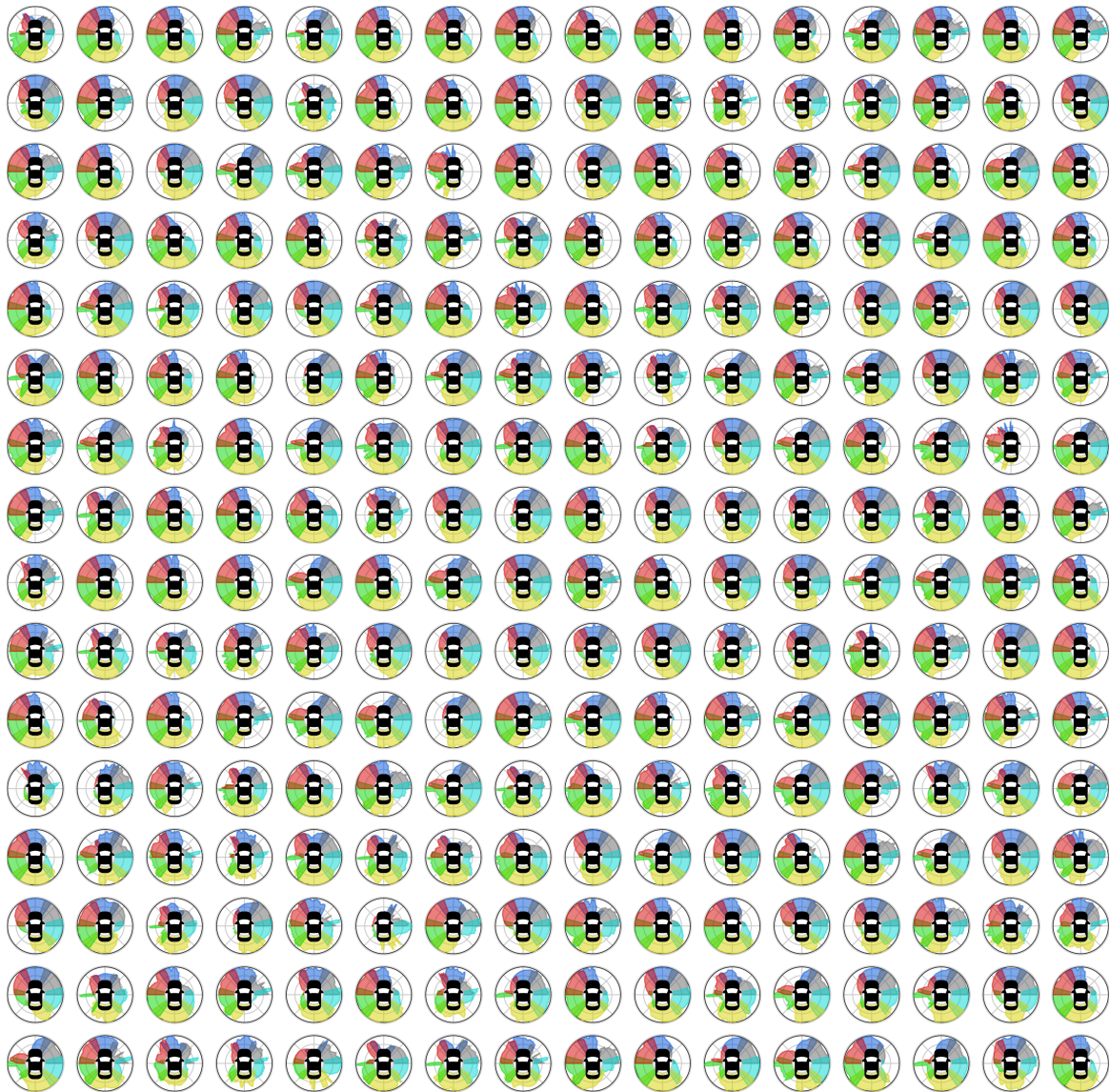


FIGURE B.3: **Input-to-latent attention study — average over heads.** These polar plots represent the directional attention intensity averaged over all attention heads for the 32 attention heads. When averaging over attention heads, we observe that the average attention spans over half of the scene. This allows latent vectors to extract long-range context between views with the capacity to disambiguate them.

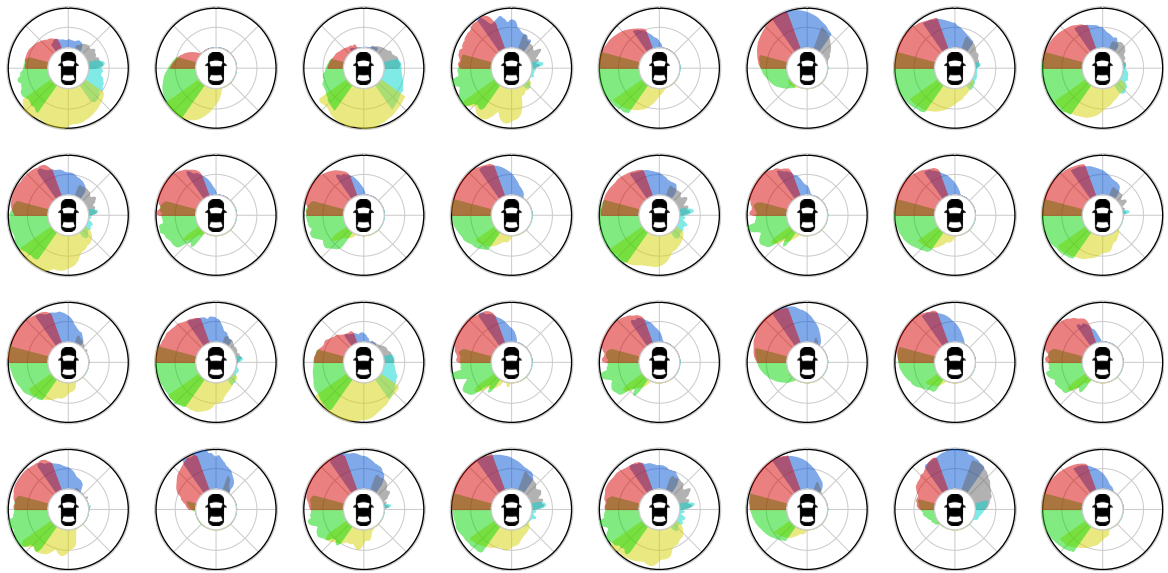


FIGURE B.4: **Input-to-latent attention study** — all the attention heads of a latent vector. These polar plots represent the directional attention intensity of the 32 attention heads for a randomly chosen latent vector (latent vector #10). As shown in [fig. B.3](#), one latent vector approximately covers half of the scene over its attention heads.



FIGURE B.5: **Input-to-latent attention study** — all the latent vectors for an attention head. These polar plots represent the directional attention intensity of the 256 latent vectors for a randomly chosen attention head (head #4). As shown in [fig. B.2](#), one attention head generally covers the full scene over the latent vectors.

Bibliography

- Amiri, A. J., Loo, S. Y., and Zhang, H. (2019). Semi-supervised monocular depth estimation with left-right consistency using deep neural network. In *IEEE ROBOTICS AND AUTOMATION*.
- Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.
- Bai, X., Hu, Z., Zhu, X., Huang, Q., Chen, Y., Fu, H., and Tai, C.-L. (2022). Transfusion: Robust LiDAR-camera fusion for 3D object detection with transformers. In *CVPR*.
- Bansal, M., Krizhevsky, A., and Ogale, A. (2018). Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst.
- Bartoccioni, F., Zablocki, E., Bursuc, A., Pérez, P., Cord, M., and Alahari, K. (2022). Lara: Latents and rays for multi-camera bird’s-eye-view semantic segmentation. In *CoRL*.
- Bartoccioni, F., Zablocki, E., Pérez, P., Cord, M., and Alahari, K. (2023). Lidartouch: Monocular metric depth estimation with a few-beam lidar. *CVIU*.
- Belkada, Y., Bertoni, L., Caristan, R., Mordan, T., and Alahi, A. (2021). Do pedestrians pay attention? eye contact detection in the wild.
- Bertozzi, M., Broggi, A., Conte, G., and Fascioli, A. (1998). Experience of the ARGO autonomous vehicle. In *Enhanced and Synthetic Vision*.
- Biasutti, P., Lepetit, V., Aujol, J.-F., Bredif, M., and Bugeau, A. (2019). Lu-net: An efficient network for 3d lidar point cloud semantic segmentation based on end-to-end-learned 3d features and u-net. In *ICCV Workshop*.
- Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.
- Boulch, A., Puy, G., and Marlet, R. (2020). FKACConv: Feature-Kernel Alignment for Point Cloud Convolution. In *15th Asian Conference on Computer Vision (ACCV 2020)*.
- Bradski, G. (2000). The opencv library. *Dr. Dobbs’s Journal of Software Tools*.
- Bryson, A., Ho, Y.-C., and Siouris, G. (1979). Applied optimal control: Optimization, estimation, and control. *Systems, Man and Cybernetics, IEEE Transactions on*, 9:366 – 367.
- Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O. (2020). nuScenes: A multimodal dataset for autonomous driving. In *CVPR*.
- Caesar, H., Kabzan, J., Tan, K. S., Fong, W. K., Wolff, E. M., Lang, A. H., Fletcher, L., Beijbom, O., and Omari, S. (2021). NuPlan: A closed-loop ML-based planning benchmark for autonomous vehicles. *arXiv 2106.11810*.

- Can, Y. B., Liniger, A., Unal, O., Paudel, D., and Van Gool, L. (2022). Understanding bird’s-eye view of road semantics using an onboard camera. *IEEE Robotics and Automation Letters*.
- Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. (2020). End-to-end object detection with transformers. In *ECCV*.
- Caron, M., Touvron, H., Misra, I., Jegou, H., Mairal, J., Bojanowski, P., and Joulin, A. (2021). Emerging properties in self-supervised vision transformers. *ICCV*.
- Casas, S., Gulino, C., Suo, S., and Urtasun, R. (2020). The importance of prior knowledge in precise multimodal prediction. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Casas, S., Sadat, A., and Urtasun, R. (2021). MP3: A unified model to map, perceive, predict and plan. In *CVPR*.
- Casser, V., Pirk, S., Mahjourian, R., and Angelova, A. (2019a). Depth prediction without the sensors: Leveraging structure for unsupervised learning from monocular videos. In *AAAI*.
- Casser, V., Pirk, S., Mahjourian, R., and Angelova, A. (2019b). Unsupervised monocular depth and ego-motion learning with structure and semantics. In *CVPR Workshop*.
- Chang, J.-R. and Chen, Y.-S. (2018). Pyramid stereo matching network. In *CVPR*.
- Chang, M., Lambert, J., Sangkloy, P., Singh, J., Bak, S., Hartnett, A., Wang, D., Carr, P., Lucey, S., Ramanan, D., and Hays, J. (2019). Argoverse: 3D tracking and forecasting with rich maps. In *CVPR*.
- Chen, K., Hong, L., Xu, H., Li, Z., and Yeung, D.-Y. (2021). Multisiam: Self-supervised multi-instance siamese representation learning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7546–7554.
- Chen, L., Sima, C., Li, Y., Zheng, Z., Xu, J., Geng, X., Li, H., He, C., Shi, J., Qiao, Y., and Yan, J. (2022). Persformer: 3d lane detection via perspective transformer and the openlane benchmark. In *European Conference on Computer Vision (ECCV)*.
- Chen, X., Ma, H., Wan, J., Li, B., and Xia, T. (2017). Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Cheng, X., Zhong, Y., Dai, Y., Ji, P., and Li, H. (2019). Noise-aware unsupervised deep lidar-stereo fusion. In *CVPR*.
- Chitta, K., Prakash, A., and Geiger, A. (2021). NEAT: Neural attention fields for end-to-end autonomous driving. In *ICCV*.
- Chitta, K., Prakash, A., Jaeger, B., Yu, Z., Renz, K., and Geiger, A. (2022). Transfuser: Imitation with transformer-based sensor fusion for autonomous driving. *IEEE TPAMI*.
- Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder–decoder approaches. *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*.

- Choy, C., Gwak, J., and Savarese, S. (2019). 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084.
- Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In *CVPR*.
- DARPA (2004). Grand challenge 2004 final report. https://www.esd.whs.mil/Portals/54/Documents/FOID/Reading%20Room/DARPA/15-F-0059_GC_2004_FINAL_RPT_7-30-2004.pdf. [Last accessed on 2023-1-2].
- Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Li, F. (2009). ImageNet: A large-scale hierarchical image database. In *CVPR*.
- Deng, Z., Todorovic, S., and Jan Latecki, L. (2017). Unsupervised object region proposals for rgb-d indoor scenes. *CVIU*, 154:127–136.
- Dickmanns, E. (2002). The development of machine vision for road vehicles in the last decade. In *IEEE Intelligent Vehicle Symposium*.
- Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*.
- Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). CARLA: An open urban driving simulator. In *CoRL*.
- Douglas, D. H. and Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: Intl. J. Geographic Information and Geovisualization*, 10(2):112–122.
- Eigen, D., Puhrsch, C., and Fergus, R. (2014). Depth map prediction from a single image using a multi-scale deep network. In *NeurIPS*.
- Elfes, A. (1990). Occupancy grids: A stochastic spatial representation for active robot perception. *UAI*.
- Ettinger, S., Cheng, S., Caine, B., Liu, C., Zhao, H., Pradhan, S., Chai, Y., Sapp, B., Qi, C. R., Zhou, Y., Yang, Z., Chouard, A., Sun, P., Ngiam, J., Vasudevan, V., McCauley, A., Shlens, J., and Anguelov, D. (2021). Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset. In *ICCV*.
- Facil, J. M., Ummenhofer, B., Zhou, H., Montesano, L., Brox, T., and Civera, J. (2019). CAM-ConvS: Camera-Aware Multi-Scale Convolutions for Single-View Depth. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Fu, H., Gong, M., Wang, C., Batmanghelich, K., and Tao, D. (2018). Deep ordinal regression network for monocular depth estimation. In *CVPR*.
- Galvani, M. (2019). History and future of driver assistance. *IEEE Instrumentation & Measurement Magazine*, 22(1):11–16.
- Gao, J., Sun, C., Zhao, H., Shen, Y., Anguelov, D., Li, C., and Schmid, C. (2020). Vectornet: Encoding hd maps and agent dynamics from vectorized representation. In *CVPR*.

- Gao, X., Hou, X., Tang, J., and Cheng, H. (2003). Complete solution classification for the perspective-three-point problem. *IEEE TPAMI*, 25:930–943.
- Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? The KITTI vision benchmark suite. In *CVPR*.
- Gidaris, S., Bursuc, A., Puy, G., Komodakis, N., Cord, M., and Perez, P. (2021). Obow: Online bag-of-visual-words generation for self-supervised learning. In *CVPR*.
- Godard, C., Aodha, O. M., Firman, M., and Brostow, G. J. (2019). Digging into self-supervised monocular depth estimation. In *ICCV*.
- Godard, C., Mac Aodha, O., and Brostow, G. J. (2017). Unsupervised monocular depth estimation with left-right consistency. In *CVPR*.
- Gong, S., Ye, X., Tan, X., Wang, J., Ding, E., Zhou, Y., and Bai, X. (2022). GitNet: Geometric prior-based transformation for birds-eye-view segmentation. *arXiv 2204.07733*.
- Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., Doersch, C., Avila Pires, B., Guo, Z., Gheshlaghi Azar, M., Piot, B., kavukcuoglu, k., Munos, R., and Valko, M. (2020). Bootstrap your own latent - a new approach to self-supervised learning. In *NeurIPS*.
- Groenendijk, R., Karaoglu, S., Gevers, T., and Mensink, T. (2020). On the benefit of adversarial training for monocular depth estimation. *CVIU*, 190:102848.
- Gruber, T., Bijelic, M., Heide, F., Ritter, W., and Dietmayer, K. (2019). Pixel-accurate depth evaluation in realistic driving scenarios. In *3DV*.
- Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies*.
- Guizilini, V., Ambrus, R., Burgard, W., and Gaidon, A. (2021). Sparse auxiliary networks for unified monocular depth prediction and completion. In *CVPR*.
- Guizilini, V., Ambrus, R., Pillai, S., Raventos, A., and Gaidon, A. (2020a). 3D packing for self-supervised monocular depth estimation. In *CVPR*.
- Guizilini, V., Hou, R., Li, J., Ambrus, R., and Gaidon, A. (2020b). Semantically-guided representation learning for self-supervised monocular depth. In *ICLR*.
- Guizilini, V., Li, J., Ambrus, R., Pillai, S., and Gaidon, A. (2019). Robust semi-supervised monocular depth estimation with reprojected distances. In *CoRL*.
- Guizilini, V., Vasiljevic, I., Ambrus, R., Shakhnarovich, G., and Gaidon, A. (2022a). Full surround monodepth from multiple cameras. *IEEE Robotics and Automation Letters*.
- Guizilini, V., Vasiljevic, I., Fang, J., Ambrus, R., Shakhnarovich, G., Walter, M., and Gaidon, A. (2022b). Depth field networks for generalizable multi-view scene representation. In *ECCV*.
- Ha, D. and Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. In *NeurIPS*.
- Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. (2023). Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*.

- Harley, A. W., Fang, Z., Li, J., Ambrus, R., and Fragkiadaki, K. (2022). Simple-BEV: What really matters for multi-sensor bev perception? *arXiv 2206.07959*.
- Harley, A. W., Li, F., Lakshmikanth, S. K., Zhou, X., Tung, H.-Y. F., and Fragkiadaki, K. (2019). Learning from unlabelled videos using contrastive predictive neural 3d mapping. In *ICLR*.
- He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. (2022). Masked autoencoders are scalable vision learners. In *CVPR*.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *CVPR*.
- Henaff, O. J., Koppula, S., Alayrac, J.-B., van den Oord, A., Vinyals, O., and Carreira, J. (2021). Efficient visual pretraining with contrastive detection. In *ICCV*.
- Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (GELUs). *arXiv 1606.08415*.
- Hendy, N., Sloan, C., Tian, F., Duan, P., Charchut, N., Xie, Y., Wang, C., and Philbin, J. (2020). FISHING net: Future inference of semantic heatmaps in grids. In *CVPR Workshop*.
- Hu, A., Corrado, G., Griffiths, N., Murez, Z., Gurau, C., Yeo, H., Kendall, A., Cipolla, R., and Shotton, J. (2022a). Model-based imitation learning for urban driving. In *NeurIPS*.
- Hu, A., Murez, Z., Mohan, N., Dudas, S., Hawke, J., Badrinarayanan, V., Cipolla, R., and Kendall, A. (2021). FIERY: Future instance segmentation in bird’s-eye view from surround monocular cameras. In *ICCV*.
- Hu, H., Liu, Z., Chitlangia, S., Agnihotri, A., and Zhao, D. (2022b). Investigating the impact of multi-lidar placement on object detection for autonomous driving. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Jacob, P., Zablocki, É., Ben-Younes, H., Chen, M., Pérez, P., and Cord, M. (2022). STEEX: steering counterfactual explanations with semantics. In *ECCV*.
- Jaegle, A., Borgeaud, S., Alayrac, J.-B., Doersch, C., Ionescu, C., Ding, D., Koppula, S., Zoran, D., Brock, A., Shelhamer, E., Henaff, O. J., Botvinick, M., Zisserman, A., Vinyals, O., and Carreira, J. (2022). Perceiver IO: A general architecture for structured inputs & outputs. In *ICLR*.
- Jaegle, A., Gimeno, F., Brock, A., Vinyals, O., Zisserman, A., and Carreira, J. (2021). Perceiver: General perception with iterative attention. In *ICML*.
- Jaritz, M., de Charette, R., Wirbel, É., Perrotton, X., and Nashashibi, F. (2018). Sparse and dense data with CNNs: Depth completion and semantic segmentation. In *3DV*.
- Jaritz, M., Vu, T.-H., de Charette, R., Wirbel, E., and Pérez, P. (2020). xMUDA: Cross-modal unsupervised domain adaptation for 3D semantic segmentation. In *CVPR*.
- Jund, P., Sweeney, C., Abdo, N., Chen, Z., and Shlens, J. (2022). Scalable scene flow from point clouds in the real world. *IEEE Robotics and Automation Letters*, 7(2):1589–1596.
- Kaushik, V., Jindgar, K., and Lall, B. (2021). Adaadepth: Adapting data augmentation and attention for self-supervised monocular depth estimation. *IEEE Robotics and Automation Letters*.

- Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J.-M., Lam, V.-D., Bewley, A., and Shah, A. (2019). Learning to drive in a day. In *ICRA*.
- Kendall, A., Martirosyan, H., Dasgupta, S., and Henry, P. (2017). End-to-end learning of geometry and context for deep stereo regression. In *ICCV*.
- Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR*.
- Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S., and Pérez, P. (2020). Deep reinforcement learning for autonomous driving: A survey.
- Koestler, L., Yang, N., Wang, R., and Cremers, D. (2020). Learning monocular 3D vehicle detection without 3D bounding box labels. In *GCPR*.
- Komodakis, N. and Gidaris, S. (2018). Unsupervised representation learning by predicting image rotations. In *ICLR*.
- Kumar, V. R., Milz, S., Witt, C., Simon, M., Amende, K., Petzold, J., Yogamani, S. K., and Pech, T. (2018). Monocular fisheye camera depth estimation using sparse lidar supervision. In *IEEE ITSC*.
- Kuznetsov, Y., Stückler, J., and Leibe, B. (2017). Semi-supervised deep learning for monocular depth map prediction. In *CVPR*.
- Lal, S., Prabhudesai, M., Mediratta, I., Harley, A. W., and Fragkiadaki, K. (2021). Coconets: Continuous contrastive 3d scene representations. In *CVPR*.
- Lee, Y. H. and Medioni, G. (2016). Rgb-d camera based wearable navigation system for the visually impaired. *CVIU*, 149:3–20.
- Lepetit, V., Moreno-Noguer, F., and Fua, P. (2009). Epnnp: An accurate o(n) solution to the pnp problem. *IJCV*, 81:155–166.
- Li, Y., Yu, A. W., Meng, T., Caine, B., Ngiam, J., Peng, D., Shen, J., Lu, Y., Zhou, D., Le, Q. V., Yuille, A. L., and Tan, M. (2022a). Deepfusion: Lidar-camera deep fusion for multi-modal 3d object detection. In *CVPR*.
- Li, Z., Wang, W., Li, H., Xie, E., Sima, C., Lu, T., Qiao, Y., and Dai, J. (2022b). BEVFormer: Learning bird’s-eye-view representation from multi-camera images via spatiotemporal transformers. *arXiv 2203.17270*.
- Lin, T.-Y., Dollar, P., Girshick, R., He, K., Hariharan, B., and Squeeze, S. (2017). Feature pyramid networks for object detection. In *CVPR*.
- Liu, R., Lehman, J., Molino, P., Such, F. P., Frank, E., Sergeev, A., and Yosinski, J. (2018). An intriguing failing of convolutional neural networks and the coordconv solution. In *NeurIPS*.
- Liu, R., Wang, J., and Zhang, B. (2020a). High definition map for automated driving: Overview and analysis. *The Journal of Navigation*.
- Liu, Y., Wang, T., Zhang, X., and Sun, J. (2022a). Petr: Position embedding transformation for multi-view 3d object detection. *arXiv*.
- Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

- Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. (2022b). A convnet for the 2020s. *CVPR*.
- Liu, Z., Wu, Z., and Toth, R. (2020b). SMOKE: Single-stage monocular 3D object detection via keypoint estimation. In *CVPR Workshop*.
- Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. (2020). Object-centric learning with slot attention. *arXiv preprint arXiv:2006.15055*.
- Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *ICLR*.
- Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *IJCV*, 60:91–110.
- Loza, A., Mihaylova, L., Canagarajah, N., and Bull, D. R. (2006). Structural similarity-based object tracking in video sequences. In *IEEE FUSION*.
- Lu, D., Xie, Q., Wei, M., Gao, K., Xu, L., and Li, J. (2022). Transformers in 3d point clouds: A survey.
- Ma, F., Cavalheiro, G. V., and Karaman, S. (2019). Self-supervised sparse-to-dense: Self-supervised depth completion from LiDAR and monocular camera. In *ICRA*.
- Ma, F. and Karaman, S. (2018). Sparse-to-dense: Depth prediction from sparse depth samples and a single image. In *ICRA*.
- Mahjourian, R., Kim, J., Chai, Y., Tan, M., Sapp, B., and Anguelov, D. (2022). Occupancy flow fields for motion forecasting in autonomous driving. *arXiv*.
- Mahjourian, R., Wicke, M., and Angelova, A. (2018). Unsupervised learning of depth and ego-motion from monocular video using 3D geometric constraints. In *CVPR*.
- Mao, J., Niu, M., Jiang, C., Liang, X., Li, Y., Ye, C., Zhang, W., Li, Z., Yu, J., Xu, C., et al. (2021). One million scenes for autonomous driving: Once dataset. *arXiv preprint arXiv:2106.11037*.
- Maturana, D. and Scherer, S. (2015). Voxnet: A 3d convolutional neural network for real-time object recognition. In *IROS*.
- Mercedes-Benz (2016). Image of the interior of the vamp3driverless car (1994). <https://group-media.mercedes-benz.com/marsMediaSite/en/instance/picture/S-Klasse-Baureihe-140-Prometheus-VaMP--Vita2.xhtml?oid=9268914>. [Last accessed on 2023-1-2].
- Mishra, A., Kumar, A., Mandloi, S., Anand, K., Zakkam, J., Sowmya, S., and Thakur, A. (2022). Evaluating and bench-marking object detection models for traffic sign and traffic light datasets. In *Proceedings of the Asian Conference on Computer Vision (ACCV) Workshops*, pages 338–353.
- Misra, I., Girdhar, R., and Joulin, A. (2021). An End-to-End Transformer Model for 3D Object Detection. In *ICCV*.
- Mohan, R. and Valada, A. (2021). EfficientPS: Efficient panoptic segmentation. *IJCV*, 129:1551 – 1579.

- Mordan, T., Cord, M., Pérez, P., and Alahi, A. (2021). Detecting 32 pedestrian attributes for autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems (T-ITS)*.
- Nayakanti, N., Al-Rfou, R., Zhou, A., Goel, K., Refaat, K. S., and Sapp, B. (2022). Wayformer: Motion forecasting via simple & efficient attention networks.
- Ng, M., Radia, K., Chen, J., Wang, D., Gog, I., and Gonzalez, J. (2020). BEV-Seg: Bird’s eye view semantic segmentation using geometry and semantic point cloud. In *CVPR*.
- NHTSA (2017). 2017 fatal motor vehicle crashes: Overview. <https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812603>. [Last accessed on 2023-01-02].
- NHTSA (2019). The Economic and Societal Impact Of Motor Vehicle Crashes, 2019 (Revised). <https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/813403>. [Last accessed on 2023-1-2].
- OECD (2017). Road safety annual report. <https://www.oecd-ilibrary.org/content/publication/irtad-2017-en>. [Last accessed on 2023-01-02].
- Pan, B., Sun, J., Leung, H. Y. T., Andonian, A., and Zhou, B. (2020). Cross-view semantic segmentation for sensing surroundings. In *IROS*.
- Park, J., Joo, K., Hu, Z., Liu, C.-K., and Kweon, I. S. (2020). Non-local spatial propagation network for depth completion. In *ECCV*.
- Phillion, J. and Fidler, S. (2020). Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3D. In *ECCV*.
- Qi, C. R., Liu, W., Wu, C., Su, H., and Guibas, L. J. (2018). Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*.
- Ravi Kumar, V., Yogamani, S., Rashed, H., Sitsu, G., Witt, C., Leang, I., Milz, S., and Mader, P. (2021). Omnidet: Surround view cameras based multi-task visual perception network for autonomous driving. *IEEE Robotics and Automation Letters*, 6(2):2830–2837.
- Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *CVPR*.
- Reiher, L., Lampe, B., and Eckstein, L. (2020). A sim2real deep learning approach for the transformation of images from multiple vehicle-mounted cameras to a semantically segmented image in bird’s eye view. In *IEEE ITSC*.
- Riegler, G., Ulusoy, A. O., and Geiger, A. (2017). Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Roddick, T. and Cipolla, R. (2020). Predicting semantic map representations from images using pyramid occupancy networks. In *CVPR*.
- Roddick, T., Kendall, A., and Cipolla, R. (2019). Orthographic feature transform for monocular 3D object detection. In *BMVC*.

- Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*.
- Saha, A., Mendez, O., Russell, C., and Bowden, R. (2021). Enabling spatio-temporal aggregation in birds-eye-view vehicle estimation. In *ICRA*.
- Sajjadi, M. S., Meyer, H., Pot, E., Bergmann, U., Greff, K., Radwan, N., Vora, S., Lucic, M., Duckworth, D., Dosovitskiy, A., Uszkoreit, J., Funkhouser, T., and Tagliasacchi, A. (2022). Scene representation transformer: Geometry-free novel view synthesis through set-latent scene representations. In *CVPR*.
- Sautier, C., Puy, G., Gidaris, S., Boulch, A., Bursuc, A., and Marlet, R. (2022). Image-to-lidar self-supervised distillation for autonomous driving data. In *CVPR*.
- Sengupta, S., Sturges, P., Ladicky, L., and Torr, P. H. S. (2012). Automatic dense visual semantic mapping from street-level imagery. In *IROS*.
- Shu, C., Yu, K., Duan, Z., and Yang, K. (2020). Feature-metric loss for self-supervised learning of depth and egomotion. *Lecture Notes in Computer Science*, page 572–588.
- Simonelli, A., Bulò, S. R., Porzi, L., Kotschieder, P., and Ricci, E. (2021). Are we missing confidence in pseudo-LiDAR methods for monocular 3D object detection? In *ICCV*.
- Sobal, V., V. J. S., Jalagam, S., Carion, N., Cho, K., and LeCun, Y. (2022). Joint embedding predictive architectures focus on slow features.
- Srikanth, S., Ansari, J. A., Ram, K., Sharma, S., Krishna Murthy, J., and Madhava Krishna, K. (2019). INFER: Intermediate representations for future prediction. In *IROS*.
- Su, H., Maji, S., Kalogerakis, E., and Learned-Miller, E. G. (2015). Multi-view convolutional neural networks for 3d shape recognition. In *ICCV*.
- Sun, P., Kretschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., et al. (2020). Scalability in perception for autonomous driving: Waymo open dataset. In *CVPR*.
- Sun, T., Segu, M., Postels, J., Wang, Y., Van Gool, L., Schiele, B., Tombari, F., and Yu, F. (2022). Shift: A synthetic driving dataset for continuous multi-task domain adaptation. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Sutton, R. S. and Barto, A. G. (1998). *Introduction to reinforcement learning*. MIT press Cambridge.
- Tan, M. and Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*.
- Tan, M., Pang, R., and Le, Q. V. (2020). Efficientdet: Scalable and efficient object detection. In *CVPR*.
- Tang, J., Tian, F.-P., Feng, W., Li, J., and Tan, P. (2020). Learning guided convolutional network for depth completion. *IEEE TIP*.
- Thomas, H., Qi, C. R., Deschaud, J.-E., Marcotegui, B., Goulette, F., and Guibas, L. J. (2019). Kpconv: Flexible and deformable convolution for point clouds. *Proceedings of the IEEE International Conference on Computer Vision*.

- Uhrig, J., Schneider, N., Schneider, L., Franke, U., Brox, T., and Geiger, A. (2017). Sparsity invariant CNNs. In *3DV*.
- Uricar, M., Krizek, P., Sistu, G., and Yogamani, S. (2019). Soilingnet: Soiling detection on automotive surround-view cameras. *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *NeurIPS*.
- Vobecky, A., Hurych, D., Siméoni, O., Gidaris, S., Bursuc, A., Pérez, P., and Sivic, J. (2022). Drive&segment: Unsupervised semantic segmentation of urban scenes via cross-modal distillation. In *arXiv preprint arXiv:2203.11160*.
- Wang, C., Buenaposada, J. M., Zhu, R., and Lucey, S. (2018). Learning depth from monocular videos using direct methods. In *CVPR*.
- Wang, H. and Tian, Y. (2022). Sequential point clouds: A survey.
- Wang, T., Zhu, X., Pang, J., and Lin, D. (2021). FCOS3D: Fully convolutional one-stage monocular 3D object detection. In *ICCV Workshop*.
- Wang, W., Dai, J., Chen, Z., Huang, Z., Li, Z., Zhu, X., Hu, X., Lu, T., Lu, L., Li, H., et al. (2022). Internimage: Exploring large-scale vision foundation models with deformable convolutions. *arXiv preprint arXiv:2211.05778*.
- Wang, Y., Chao, W.-L., Garg, D., Hariharan, B., Campbell, M., and Weinberger, K. (2019). Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *CVPR*.
- Watson, J., Firman, M., Brostow, G. J., and Turmukhambetov, D. (2019). Self-supervised monocular depth hints. In *ICCV*.
- WHO (2016). Global status report on road safety 2018. <https://www.who.int/publications/i/item/9789241565684>. [Last accessed on 2023-01-02].
- Wimbauer, F., Yang, N., Rupperecht, C., and Cremers, D. (2023). Behind the scenes: Density fields for single view reconstruction. *arXiv preprint arXiv:2301.07668*.
- Xiong, Y., Ren, M., Zeng, W., and Waabi, R. U. (2021). Self-supervised representation learning from flow equivariance. In *ICCV*.
- Xu, J., Zhang, R., Dou, J., Zhu, Y., Sun, J., and Pu, S. (2021). Rpvnet: A deep and efficient range-point-voxel fusion network for lidar point cloud segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16024–16033.
- Xu, Y., Zhu, X., Shi, J., Zhang, G., Bao, H., and Li, H. (2019). Depth completion from sparse LiDAR data with depth-normal constraints. In *ICCV*.
- Yamanaka, K., Matsumoto, R., Takahashi, K., and Fujii, T. (2020). Adversarial patch attacks on monocular depth estimation networks. *IEEE Access*, 8.
- Yang, G. and Ramanan, D. (2020). Upgrading optical flow to 3d scene flow through optical expansion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

- Ye, W., Liu, S., Kurutach, T., Abbeel, P., and Gao, Y. (2021). Mastering atari games with limited data. In *NeurIPS*.
- Yifan, W., Doersch, C., Arandjelović, R., Carreira, J., and Zisserman, A. (2022). Input-level inductive biases for 3D reconstruction. In *CVPR*.
- Yin, Z. and Shi, J. (2018). GeoNet: Unsupervised learning of dense depth, optical flow and camera pose. In *CVPR*.
- Yogamani, S., Hughes, C., Horgan, J., Sistu, G., Varley, P., O’Dea, D., Uricar, M., Milz, S., Simon, M., Amende, K., Witt, C., Rashed, H., Chennupati, S., Nayak, S., Mansoor, S., Perrotton, X., and Perez, P. (2019). Woodscape: A multi-task, multi-camera fisheye dataset for autonomous driving. In *ICCV*.
- Zablocki, É., Ben-Younes, H., Pérez, P., and Cord, M. (2022). Explainability of deep vision-based autonomous driving systems: Review and challenges. *IJCV*.
- Zemni, M., Chen, M., Zablocki, É., Ben-Younes, H., Pérez, P., and Cord, M. (2023). OCTET: object-aware counterfactual explanations. In *CVPR*.
- Zeng, W., Luo, W., Suo, S., Sadat, A., Yang, B., Casas, S., and Urtasun, R. (2019). End-to-end interpretable neural motion planner. In *CVPR*.
- Zhao, C., Zhang, Y., Poggi, M., Tosi, F., Guo, X., Zhu, Z., Huang, G., Tang, Y., and Mattoccia, S. (2022). Monovit: Self-supervised monocular depth estimation with a vision transformer. In *3DV*.
- Zhao, H., Gao, J., Lan, T., Sun, C., Sapp, B., Varadarajan, B., Shen, Y., Shen, Y., Chai, Y., Schmid, C., Li, C., and Anguelov, D. (2020). Tnt: Target-driven trajectory prediction.
- Zhao, S., Gong, M., Fu, H., and Tao, D. (2021). Adaptive context-aware multi-modal network for depth completion. *IEEE TIP*.
- Zhou, B. and Krähenbühl, P. (2022). Cross-view transformers for real-time map-view semantic segmentation. In *CVPR*.
- Zhou, B., Krähenbühl, P., and Koltun, V. (2019). Does computer vision matter for action? *Science Robotics*.
- Zhou, T., Brown, M., Snavely, N., and Lowe, D. G. (2017). Unsupervised learning of depth and ego-motion from video. In *CVPR*.
- Zhu, X., Su, W., Lu, L., Li, B., Wang, X., and Dai, J. (2021). Deformable DETR: Deformable transformers for end-to-end object detection. In *ICLR*.
- Zhu, X., Yin, Z., Shi, J., Li, H., and Lin, D. (2018). Generative adversarial frontal view to bird view synthesis. In *3DV*.