SCUOLA DI DOTTORATO
UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA

Department of Computer Sciences, Systems and Communications
PhD program in Computer Science Cycle XXXII

Machine Learning Techniques for Urban Vehicle Localization

Tutor: Prof. Giuseppe VIZZARI

Supervisor: Prof. Domenico G. SORRENTI

Coordinator: Prof. Leonardo MARIANI

PhD dissertation by:

Daniele CATTANEO

Registration number 735879

Academic Year 2018-2019

Michael - *Should I say thanks?*
KITT - *If you do I'll say "You're welcome"*
Michael - *Thanks*
KITT - *De nada*

Contents

Chapter 1

Introduction

It was almost one century ago, in 1925, that Houdina Radio Control for the first time envisioned and demonstrated a car without a driver going trough the streets of New York City. Although the car was not autonomous, but was instead radio-controlled, the idea of releasing drivers from tedious tasks was already clear. It was only 60 years later, during the 1980s, that the first experiments on self-driving cars were pursued by Eric Dickmanns and his team. In 1987, Dickmanns' car was the first ever vehicle to drive autonomously on an empty highway. Since then, the interest in self-driving cars increased from year to year, especially after the *Grand Challenges* organized by the Defense Advanced Research Projects Agency (DARPA) between the years 2004 and 2007. Many research teams all over the world competed in these challenges in order to win the first prize. Nowadays, most of the big car-making companies are investing many resources on autonomous car related research, and semi-autonomous vehicles are already becoming available on the market. The World Health Organization estimated that car accidents cause 1.35 million death every year, and that it is the most common cause of death in children and young adults [1]. One of the primary aims of the next generation of Advanced Driver Assistance Systems (ADAS) and of self-driving cars is to provide safe systems that will drastically reduce the number of these deaths. Beside safety, self-driving cars can provide personal mobility to people that are unable to drive due to disability. Finally, autonomous vehicles can regulate their speed in order to reduce fuel consumption.

1.1 Motivations

Robots that operate in controlled environments, such as factories, are usually programmed to perform only specific tasks, and the unpredictability factors in such environments are very limited. For examples, robots for welding, assembling and painting that can be found in car manufacturing factories are usually enclosed in cages. Therefore, they don't usually exploit any algorithms for understanding the surrounding scene. Self-driving cars and other robots that navigate and interact with dynamic environments, on the other hand, require a complete understanding of their surrounding in order to deal with our complex world. Since these robots coexist with human actors (*e.g.,* drivers and pedestrians), their reliability is of primary importance. An erroneous detection of the perception system, or a wrong decision, might lead to accidents and even fatal injuries for other road users.

Beside the low-level control of the vehicle, the three main tasks for a self-driving car are the *perception*, the *localization*, and the *motion planning*. In the perception, the sensors' data are collected and processed to understand the surrounding scene. Methods that fall in this category aim, for example, at object detection and tracking to avoid collision, detecting the space where the vehicle can drive, crosswalk detection in order to stop if pedestrians are crossing the street, pedestrians' intention prediction so to slow down if a child is running on the sidewalk. The localization task aims at finding the position and orientation of the vehicle with respect to a reference map. Such map can be of different types, *e.g.,* road graph, geometric map or grid map. It is important to note that Global Navigation Satellite Systems (GNSSs), such as Global Positioning System (GPS), GLONASS, Galileo, and BeiDou, are not accurate and reliable enough for autonomous driving applications, especially in urban environments, where buildings may block or deflect satellites' signals, leading to non-line-of-sight (NLOS) or multipath errors [2]. Therefore, to overcome the latter issues, localization approaches usually exploit perception techniques to match the perceived scene with the map. For example, the localization can be performed by reconstructing the surrounding geometry and looking for the same geometry within the map. In the motion planning task, both perception and localization are exploited in order to take decisions such as which road to take, or whether to overtake other cars.

During the last decade, the interest in Artificial Neural Networks (ANNs) has drastically increased from both academia and industry. Deep learning based techniques have

been successfully applied in many different fields, such as image classification [3], medical image segmentation [4], natural language processing [5], and even for playing Go [6]. Nowadays, also in the autonomous driving field, state-of-the-art approaches that tackle the perception task usually exploit Convolutional Neural Networks (CNNs). For examples, CNN-based methods have outperformed previous techniques in object detection [7–9], pixel-level semantic classification [10–12], geometric reconstruction from cameras [13–16], and optical flow estimation [17–20]. Recently, Deep Neural Networks (DNNs) have also been used for directly localizing a robot in an end-to-end fashion, *i.e.,* combining both the perception and the localization tasks in a single step [21–23]. However, these approaches require a huge amount of images (and relative positions) of the environment to train the model, and second, they achieved good performances only in small environments (*e.g.,* a room or a small outdoor area of few tens of meters). Moreover, DNNs for localization can not be used to obtain a localization in areas different from the one represented in the training set, since these network learn to perform localization only on the specific scene used during the training phase. In this thesis, we present innovative methods to effectively exploit DNNs and machine learning techniques for autonomous vehicles localization.

1.2 Problem Statement

Over the past few years, the effectiveness of scene understanding for self-driving cars has drastically increased, mainly thanks to the great performances achieved by CNNs and the great interest in self-driving cars related topics from government agencies and big companies. Even though these improvements allowed for more advanced and sophisticated ADAS and maneuvers, the current state of the art is far from the SAE full-automation level[1], especially in complex scenarios such as urban areas. Most of these algorithms depend on very accurate localization estimates. A typical localization pipeline usually involves two steps. First, a rough pose of the robot is estimated, thus performing a so-called *global* localization of the observer. Secondly, the initial rough pose is refined, achieving more precise localization accuracy level, usually referred to as *local* localization. Usually, the first step can be accomplished with the aid of GNSSs, which provides a global position with respect to a World Geodetic System (WGS). However, the local-

[1]The SAE International "Levels of Driving Automation" standard define six different levels of automation, ranging from manual to fully automated systems.

ization accuracy and reliability of GNSSs is inadequate for the second step, particularly in urban environments, where buildings might obstruct satellites' signal, causing NLOS and multi-path issues. Therefore, for the second step, complementary approaches are required. Among these complementary systems, in the robotic and computer vision communities, a common way is to exploit scene understanding techniques.

Different options have been investigated to solve the local localization problem, including approaches based on both vision and LiDAR; they usually exploit an a-priori knowledge of the environment (a map) in the localization process. Although LiDAR sensors are the de facto standard for 3D geometric reconstruction, their price, weight, and mechanically-based scanning systems, make them still not suitable for mass installation on cars. Camera sensors, on the other hand, are cheaper and weight less than LiDAR, but the geometric reconstruction that can be obtained with cameras (in both single and stereo configurations) is not as accurate as the one that can be obtained with LiDAR. Moreover, the camera reconstruction errors are not linear, but instead increase quadratically with the distance from the observer. While approaches that localize a vehicle using LiDAR sensors achieved accuracies in the order of 10 cm [24], camera-only localization techniques are still far from this level of localization in city-scale complex environments.

Recalling that localization is the task aimed at finding the position and the orientation of a robot with respect to a reference map, different types of representation can be used for this map. For examples, grid maps (bi-dimensional or tri-dimensional) are a type of maps commonly used in controlled environments. This type of maps divides the world into a grid of fixed size and, in its basic form, store in each cell a binary value that represent whether that cell is traversable or occupied. Point clouds maps, instead, represent the environment as a simple list of 3D points. Another possibility is to exploit maps provided by map-making companies, such as HERE, TomTom or the collaborative project OpenStreetMap (OSM). These maps can be seen as a graph, where vertices represent geo-referenced points, and edges represent roads that connect the associated points. Finally, although not yet freely available, map-making companies are already developing the so-called HD-maps, that are specifically designed to support autonomous driving applications. These maps provide an accurate position of high-level features such as traffic lights and road lanes as well as a geometric representation of the environment.

In this thesis, we address the vehicle localization task in challenging environments, using only cheap camera sensors onboard. Our goal is to take advantage of recent DNNs

and machine learning techniques to localize a vehicle with respect to maps provided by map-making companies (cartographic and HD-maps). Since HD-maps are not yet freely available, we resemble them using maps built from LiDAR sensors. In particular, we try to answer the following questions:

- Which types of maps are better suited for vision-based localization of a self-driving car?
- How to integrate existing CNNs and machine learning techniques within a local localization pipeline?
- How can we provide an accurate localization near road intersections, one of the most dangerous situations for vehicles?
- Can we leverage buildings information to enhance vehicle localization in urban environments?
- Is it possible to design a DNN that directly localize the camera and that can be used in any environment (even areas not seen in the training phase) without re-training?
- How can we globally localize a vehicle within a city when GNSSs are not available?

1.3 Scientific Contributions

In the first part of this thesis (Chapters 3 to 5) we propose three approaches that exploit data from the OSM service to localize a vehicle in highway-like and urban environments by matching high-level features (road lanes, road intersections and buildings), integrating CNNs and machine learning in different localization pipelines. In the second part (Chapters 6 and 7), instead, we exploit LiDAR-maps (the same data that will probably be available in the forthcoming market of HD-maps), and we propose novel DNN-based systems that directly perform localization in an end-to-end fashion.

The research proposed in this thesis is mainly focused on the local localization task, *i.e.,* a rough position estimate is required to be known a priori. The reason behind this decision is that such a rough position estimate can be usually obtained with GNSSs. However, to be able to localize the vehicle even when GNSSs are not available, in Chapter 7 we complete our contribution proposing a novel global localization method.

The scientific contributions of each approach proposed in this thesis can be summarized as follows.

- In Chapter 3 we proposed an approach to estimate the ego-lane of the vehicle, *i.e.,* to answer the question *"in which lane am I driving?"*, in highway-like scenarios. We tackle the problem in a probabilistic fashion by exploiting a Hidden Markov Model (HMM). The method relies on existing line detectors and trackers, and aims to improve the ego-lane estimates that can be obtained only from the detector's output. Typical line detectors often fail to detect all the road lines due to, for example, shadows or traffic occlusions. Therefore, the proposed HMM explicitly includes a variable that represent the functioning status of the detector itself.

- In Chapter 4 we introduce a method to localize a vehicle in complex urban environments, specifically designed for localization in proximity of road intersection areas. Here state-of-the-art CNNs and visual odometry techniques are combined in order to generate an occupancy grid that represents the geometry of the upcoming intersection. The localization is performed by comparing the reconstructed geometry with its counterpart in the OSM service. Different localization hypotheses and geometries are taken into account, to be robust to small errors in the OSM data.

- In Chapter 5 we propose to exploit one of the most static structures in urban environments, *i.e.,* the buildings, in the vehicle's surrounding to enhance the vehicle localization. The buildings' façades detected from the cameras are matched with the buildings' outlines gathered from OSM to improve the localization when buildings are visible. State-of-the-art CNNs are exploited to accurately perceive and reconstruct the surrounding buildings.

- In Chapter 6 we address the problem of monocular camera local localization in HD-maps. The different nature of the sensors used for localization (camera) and for mapping (LiDAR) makes this task extremely challenging. To address this problem, we propose a CNN, named CMRNet, that learns how to match an image with a LiDAR-map. Since the network does not learn the map, it can be used in any environment for which a LiDAR-map is available, without the need to retrain the network.

- While all the latter approaches addressed the local localization task (*i.e.,* they require a rough position estimate to be known), in Chapter 7 we addressed the

global visual localization task. In particular, we want to localize an image in an HD-map without any prior information about its position. This is useful in case where GNSSs are not available. To address this task, we propose to jointly train two DNNs, one for the images and the other for the point clouds, in order to generate a shared embedding space for the two types of data. This embedding space allows us to perform place recognition with heterogeneous sensors.

Moreover, since we were unable to find suitable datasets for testing the approaches presented in Chapters 3 and 4, we published two new datasets.

- ego-lane1 dataset: we recorded a novel dataset composed of more than 100 km of highway driving in both Italy and Spain. We manually labelled each frame with the correspondent vehicle's ego-lane (*i.e.,* in which lane the vehicle is driving). This dataset was recorded in wide highways (three and four lanes), and include more than 100 lane changes. This dataset is available online[2].

- intersection dataset: we performed an extensive analysis of the available datasets for autonomous driving to assemble a new dataset, specially designed for benchmarking localization at intersections. After the analysis performed, we ended up with 48 approaches to intersections sequences, all from KITTI residential category. Although all approaches are from the same German city, we were able to include different intersection geometries, lighting and traffic conditions. This dataset is available online[3].

1.4 Publications

The majority of the approaches presented in this thesis have been published in peer-reviewed conferences' proceedings. The publications related to the research presented in this thesis are listed below.

- A. L. Ballardini, D. Cattaneo, R. Izquierdo, I. Parra, M. A. Sotelo, and D. G. Sorrenti. "Ego-lane estimation by modeling lanes and sensor failures". In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. Oct. 2017, pp. 1–7

[2] http://www.ira.disco.unimib.it/ego-lane-estimation-by-modeling-lanes-and-sensor-failures
[3] http://www.ira.disco.unimib.it/visual-localization-at-intersections-with-digital-maps

- A. L. Ballardini, D. Cattaneo, S. Fontana, and D. G. Sorrenti. "An online probabilistic road intersection detector". In: *IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 239–246

- A. L. Ballardini, D. Cattaneo, and D. G. Sorrenti. "Visual Localization at Intersections with Digital Maps". In: *IEEE International Conference on Robotics and Automation (ICRA)*. May 2019, pp. 6651–6657

- A. L. Ballardini, D. Cattaneo, S. Fontana, and D. G. Sorrenti. "Leveraging the OSM building data to enhance the localization of an urban vehicle". In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. Nov. 2016, pp. 622–628

- D. Cattaneo, M. Vaghi, A. L. Ballardini, S. Fontana, D. G. Sorrenti, and W. Burgard. "CMRNet: Camera to LiDAR-Map Registration". In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. Oct. 2019, pp. 1283–1289

- D. Cattaneo, M. Vaghi, S. Fontana, A. L. Ballardini, and D. G. Sorrenti. "Global visual localization in LiDAR-maps through shared 2D-3D embedding space". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020. Submitted

1.5 Thesis Outline

This thesis is organized as follows. In Chapter 2 we introduce a brief historical preface and background knowledge on the topics of self-driving cars, robot localization and artificial neural networks. In Chapter 3 we propose a probabilistic model for estimating the vehicle's ego-lane in highway-like scenarios. Chapter 4 introduces a localization pipeline specifically designed for road intersection detection that exploits state-of-the-art CNNs and the OSM road graph. In Chapter 5 we present a method to enhance the vehicle's localization by leveraging buildings information from the OSM service. Chapter 6 proposes an end-to-end CNN for monocular camera localization that can be used in any environments for which a LiDAR-map is available, while in Chapter 7 we introduce a DNN-based system for visual global localization in LiDAR maps. Finally, in Chapter 8 we draw our conclusions about the research presented in this thesis.

Chapter 2

Background

2.1 Autonomous Driving

The first concept of a self-driving car dates back to 1925 when Houdina Radio Control demonstrated the "American Wonder", a vehicle that was radio-controlled trough New York City traffic. A decade later, in 1939, General Motors sponsored an exhibit called "Futurama" designed by Norman Bel Geddes. The exhibit was a vision 20 years into the future depicting automated highways for traffic congestion reduction. Circuits embedded in the roadways powered radio-controlled electric cars. Even though this generation of vehicles was not really autonomous, the idea of releasing the driver from tedious tasks was already evident.

In 1977, thanks to the advancements in computer vision tasks and the higher computational power available, the Tsukuba Mechanical Engineering Lab unveiled the first truly autonomous car. It was equipped with two cameras and an analog computer that could track white street markers, allowing the car to drive up to 30 km/h in a dedicated road. A few years later, Daimler in collaboration with Ernst Dickmanns and his team equipped a van (named VaMoRs) with two cameras and multiple microprocessors [25]. VaMoRs was able to drive at speeds up to 96 km/h on empty highways. Afterwards, the EUREKA organization funded the "Eureka Prometheus" project, the largest research and development program on self-driving cars. The project culminated in 1995, when Dickmanns' autonomous vehicle took a 1600 km trip from Munich to Copenhagen and back almost fully autonomously, with only 5% of human intervention. The car drove at a speed up to 175 km/h in standard traffic conditions, autonomously overtaking other cars. Around the same years, the NAVLab project from Carnegie Mellon University (CMU) tested a series of autonomous vehicles. Notably, in 1995, NAVLab 5 drove 5000 km with

autonomous steering controls (for about 98% of the journey) but manual throttle and brake controls [26]. In 1996, professor Alberto Broggi and his team launched the project ARGO [27]. Their vehicle completed a 1900 km journey in Italian highways, of which 94% was autonomously controlled, with an average speed of 90 km/h. This second generation of vehicles was really autonomous but was focused on simplistic scenarios, *i.e.,* dedicated roads or highways with clear lane markings.

In 2004, the DARPA introduced "Grand Challenge", the first long-distance competition for self-driving cars. International teams were challenged to build a vehicle that could drive autonomously along a 240 km route in the Mojave Desert region. However, the farthest distance traveled was only 11 km. A year later, the second round of the Grand Challenge took place, and this time five teams were able to complete the 212 km route. The winning vehicle, Stanley [28] developed by the Stanford Racing Team, used multiple LiDAR units, a camera and probabilistic reasoning for detecting road surface and obstacles. In 2007, DARPA conducted a third competition, called "Urban Challenge", in which self-driving cars were demanded to handle city-like environments obeying traffic regulations and negotiation intersections with other vehicles. Six teams were able to complete the route, and the winning vehicle Boss, developed at Carnegie Mellon University [29], uses multiple LiDAR, radars, camera and GPS units to handle the complex environment. The DARPA challenges tackled complex scenarios (desert roads and urban settings), but in controlled environments, *i.e.,* no pedestrians or bicycles and lanes wider than usual.

In 2009, Google recruited experienced researchers from the best teams involved in the DARPA Grand Challenges and started his self-driving car project. In 2016 Google states that his self-driving cars drove more than three million kilometers in autonomous mode, though they were involved in 14 minor accidents caused by humans driving other vehicles. During the last decade, most of the big car making companies (such as BMW, Toyota, General Motors, Bosch, Audi, Mercedes), AI companies (Google, Apple) and innovative companies (Uber, nuTonomy, Tesla Motors) invested lots of resources in research and development of self-driving cars. Since each company has different name and technologies for the same capabilities, in 2014 "SAE International" introduced the J3016 document "Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems". This document defined six different levels of automation, ranging from manual to fully automated systems, based on the human intervention and attention required.

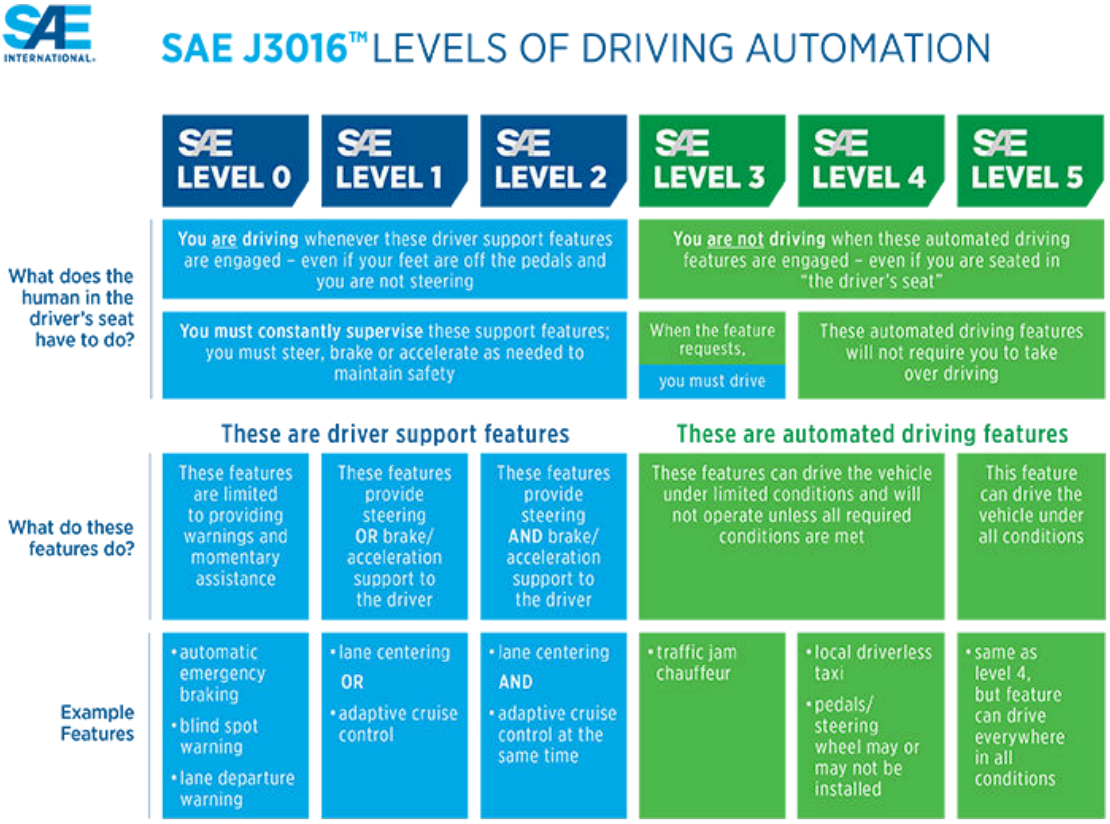


Figure 2.1: The SAE International "Levels of Driving Automation" standard.

Figure 2.1 reports the descriptions of the different automation levels defined in the document.

Nowadays, cars with semi-autonomous features (SAE level 2) are already available on the market. Most of the big car making companies (such as Audi, BMW, General Motors, Ford, Honda, Hyundai, Lexus, Mercedes-Benz, Nissan, Porsche, Tesla, Toyota and more) offer cars equipped with a combination of adaptive cruise control and lane-keeping to autonomously drive inside a lane in highways. Some of these companies also offer more advanced systems:

- Tesla Motors' Autopilot: the first and probably the most advanced semi-autonomous system on the market, it features adaptive cruise control, lane-keeping, auto lane change, auto-park and interchanges navigation. The system requires the driver to pay attention and keeps his hands on the steering wheel.

- General Motors' SuperCruise: this system offers fewer features than Autopilot (only adaptive cruise control and lane-keeping on selected highways), but it is the first "True Hand-Free" system[1]. The driver does not need to keep his hands on the steering wheel, but needs to pay attention (a camera tracks the eye of the driver to make sure he is paying attention).
- BMW' Extended Traffic Jam Assistant: similarly to SuperCruise, it offers hand-free driving for speeds up to 60 km/h

Moreover, Audi promised to release the first SAE level 3 (driver does not need to pay attention, but need to take control in certain situations) feature on the market. However, this feature will only work in traffic jams up to 60 km/h on highways where a physical barrier separates the two carriageways.

All the aforementioned modern self-driving cars either work only on highways or rely on expensive sensors such as LiDAR units. In this thesis, instead, we focus on more challenging scenarios (urban areas) using only cheap onboard sensors (cameras).

2.2 Localization

The localization is one of the most important tasks for an autonomous mobile robot, and the research community have tackled it using different approaches. Being able to localize itself is crucial for an autonomous robot as the collision avoidance with static or dynamic objects, and the prediction of a proper path to the goal destination, strictly depend on this capability. More formally, localization is the task aimed at finding the position and orientation of the robot with respect to a known reference map of the working environment. The combination of position and orientation is also referred to as *pose*.

GNSSs (such as GPS, Galileo, GLONASS and BeiDou) provide a global localization anywhere on the Earth, provided that there is a clear line of sight to at least four satellites. In real-world autonomous driving applications, however, this is not often the case: in tunnels, for example, such systems are not available. Even in outdoor environments GNSSs are often inadequate for self-driving cars, especially in urban environments where buildings may block or deflect satellites' signals, leading to non-line-of-sight (NLOS) or multipath errors [2]. Therefore, GNSSs by themselves are not accurate and reliable

[1] https://www.cadillac.com/ownership/vehicle-technology/super-cruise

enough for autonomous driving applications. Different approaches for mitigating the latter issues have been proposed, for example by exploiting filtering techniques [30, 31] or by using three-dimensional mapping of the buildings [32, 33]. However, the obtained localization is still not accurate enough (*i.e.,* they reduced the localization errors from tens of meters to 5-10 meters). A common practice in the autonomous driving community is to exploit vehicles' onboard sensors in order to obtain localization estimates appropriate for safe navigation.

A typical localization pipeline usually involves two consecutive steps: first, the sensors' data are processed to detect some informative features and second, those features are matched with the reference map. Numerous approaches have been proposed by the research community, and they differ in the type of sensor used, type of the map and working environment (*i.e.,* indoor or outdoor). Another important distinction is whether the approach requires an initial position estimate to be known ("local" localization) or not ("global" localization). Global techniques estimate a rough position of the robot globally within the reference map without any prior information about its position, while Local techniques aim at refining such rough pose to provide an accurate localization.

In the following subsections, we will present an overview of the existing types of map used for localization in the context of autonomous driving.

2.2.1 Point Cloud Maps

The first type of maps we introduce is the point cloud maps in which a simple list of points represent the environment. These points, besides their spatial coordinates, can optionally have additional information attached, such as intensity, color or surface normal. On the one hand, this type of map is usually not discretized, and thus, it is the most expressive among the types presented in this overview. In fact, other types of maps, such as grid maps (Section 2.2.2), are usually generated starting from point clouds. On the other hand, point cloud maps usually require lot of memory to be stored, and they do not explicitly distinguish between free and unknown space.

Point clouds maps are usually generated using LiDAR sensors [34, 35], thanks to their high accuracy in measuring distances and thus producing accurate maps. However, various approaches to generate point cloud maps with different sensors have been proposed, *e.g.,* using stereo cameras [36], depth cameras [37] or even monocular camera [38]. Point cloud maps generated with stereo or depth cameras are usually denser than LiDAR maps

but produce a less accurate reconstruction.

In order to reduce the memory required to store point cloud maps, a common approach is to "subsample" the map, that means keeping only a subset of the points. Besides the most straightforward strategy to remove random points, more sophisticated strategies have been proposed to select the points to remove based, for example, on clustering [39] or points distances [40] approaches.

Point cloud maps are usually generated through a family of approaches that takes the name of Simultaneous Localization And Mapping (SLAM). The mapping process of a SLAM approach typically involves a mobile robot that navigate in an unknown environment, progressively mapping the area and at the same time localizing within it. Another relevant and widespread technique for generating point cloud maps is called Structure from Motion (SfM) [41, 42]. While SLAM is generally sensor agnostic, SfM is specific for camera sensors. In a typical SfM pipeline, multiple images of the environment without known poses are processed, generating a sparse point cloud. In the first step, features such as SIFT [43] or SURF [44] are detected in all the images. Afterword, these features are tracked between overlapped images. Finally, both camera poses and 3D reconstruction are estimated employing a bundle adjustment technique [45]. However, SfM applied to monocular systems is subject to the scale ambiguity, *i.e.,* camera poses and point cloud map are estimated up to an unknown scale factor. In order to estimate this scale factor, either relative motion between camera poses or dimension of a scene object needs to be known.

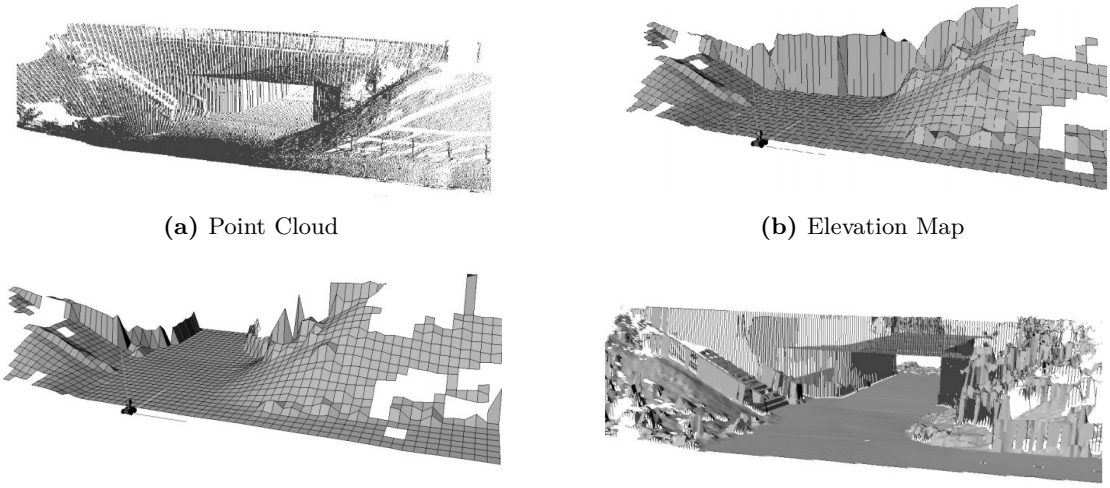
2.2.2 Grid Maps

This type of maps divides the world into a grid of fixed size, and store in each cell a value that represents the probability of that cell to be occupied. Grid maps, despite their simplicity, are still widely used for localization, especially in indoor environments where a 3-DoF localization is enough. For this purpose, bi-dimensional grids depicting the working environment in a *bird's-eye view (BEV)* perspective are used. Approaches from the late 1990s and still commonly used today [46, 47] use bi-dimensional grid maps, allowing for robot localization by matching LiDAR scans with the map. An example of a map generated by those approaches is depicted in Figure 2.2. These BEV grid maps require less memory and computational power than point cloud maps, but they only work in environments with a flat ground (*e.g.,* buildings) and ground robots.

Figure 2.2: A bi-dimensional grid map of the environment where gray pixels represent free space, black pixels represent obstacles and teal pixels represent unknown areas. The LiDAR scan at the estimated robot position is shown in blue. The good alignment of the measures with respect to the map indicates a good localization estimate.

In order to deal with non-flat terrains, a popular approach is to use elevation maps (also known as 2.5D maps). This type of map uses a BEV grid wherein each cell is stored the terrain's height of the corresponding world space. Some researchers have tested elevation maps in a planetary explorer rover for mapping the environment [48] and localization of the robot [49]. However, standard elevation maps cannot properly represent vertical or overhanging objects, such as walls of buildings, bridges and trees' branches. Different extensions have been proposed to deal with the problem mentioned above. Extended Elevation Maps [50] can detect vertical and overhanging objects, but they can represent only one surface per cell. On the other hand, Multi-Level Surface Maps [51] store multiple surfaces in each cell of the grid, correctly handling situations like bridges. Figure 2.3 depicts an example of elevation maps and their extensions.

Grid-based maps can also be used with three-dimensional grids, where each cell stores



(a) Point Cloud

(b) Elevation Map

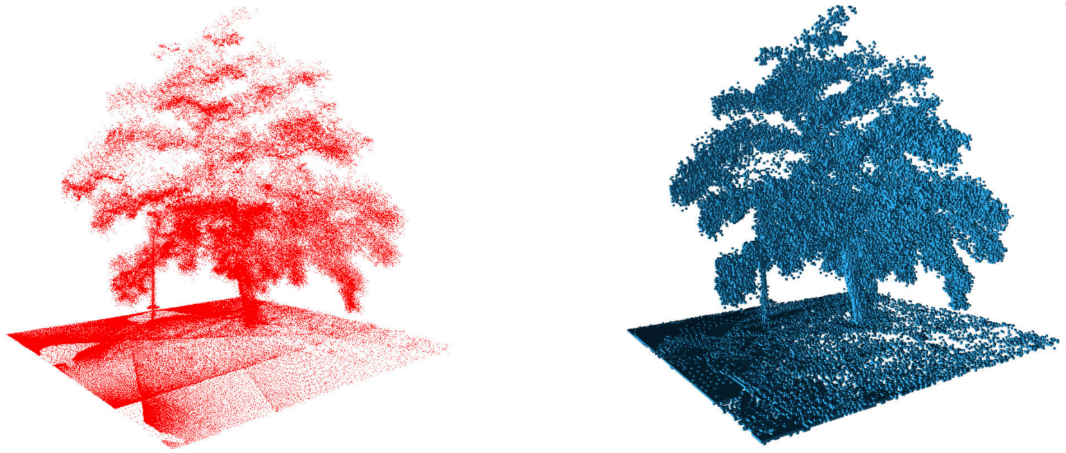
(c) Extended Elevation Map

(d) Multi-level surface map

Figure 2.3: How different elevation maps handle the situation in **(a)**: standard elevation maps **(b)** see an impassable wall, while extended elevation maps **(c)** represent the ground of the underpass and multi-level surface maps **(d)** correctly depict both the surfaces under and over the bridge. Images taken from [51].

the state (free or occupied) of the corresponding environment space. In this case, the cells are called voxels, from volumetric pixels. These maps are more expressive than 2D and 2.5D grid maps but require more memory. It is important to note that all the grid-based maps mentioned so far require that all the map must be allocated beforehand, and the maximum extension must be known a priori.

A relevant extension to the standard 3D grid maps, called **OctoMap** [52], offer different advantages such as low memory requirement, explicit representation of free and unknown space, multi-resolution queries and probabilistic measurements' integration. OctoMaps are based on *octrees*, a hierarchical data structure for 3D geometric modeling [53]. Each node of the tree represents a voxel that can be recursively subdivided into eight sub-nodes until reaching the minimum voxel dimension defined. The hierarchical nature of the data structure allows for queries at multiple resolutions. Moreover, OctoMaps are dynamically expanded as needed; thus, the maximum extension does not need to be known beforehand. An example of an OctoMap is depicted in Figure 2.4.



(a) Point Cloud **(b)** OctoMap

Figure 2.4: An example of an OctoMap (b) generated starting from the point cloud in (a). Images taken from [52].

2.2.3 Topological Maps

The last type of map that can be exploited for autonomous robot localization is the topological map. These maps were not initially intended for robotic applications, but were instead intended for human reading, and later for navigation applications that can be found in any modern smartphone. The most basic form of topological maps can be seen as a graph, where vertices represent geo-referenced points, and edges represent roads that connect the associated points. The edges of the graph can also contain additional information, such as name, type and width of the road, number of lanes and maximum speed allowed. Different companies, such as TomTom, HERE and Google, provide these maps, but also free and collaborative projects provide them. In 2004, Steve Coast, inspired by the success of Wikipedia, founded OpenStreetMap (OSM), a collaborative project to create editable maps of the world that are freely available to the public [54]. An example of the data from OSM is depicted in Figure 2.5.

Although OSM is usually generated by gathering GPS data manually collected by users, some researchers have proposed different methods to automatically correct or enhance the map. Aerial images can be used to automatically extract road-segment center-lines [55, 56], roads' width [57] or coupled with ground images to enrich the map with segmentation categories such as sidewalks and parking slots [58]. OSM does not only provide the road network, but it also contains additional semantic information, such as

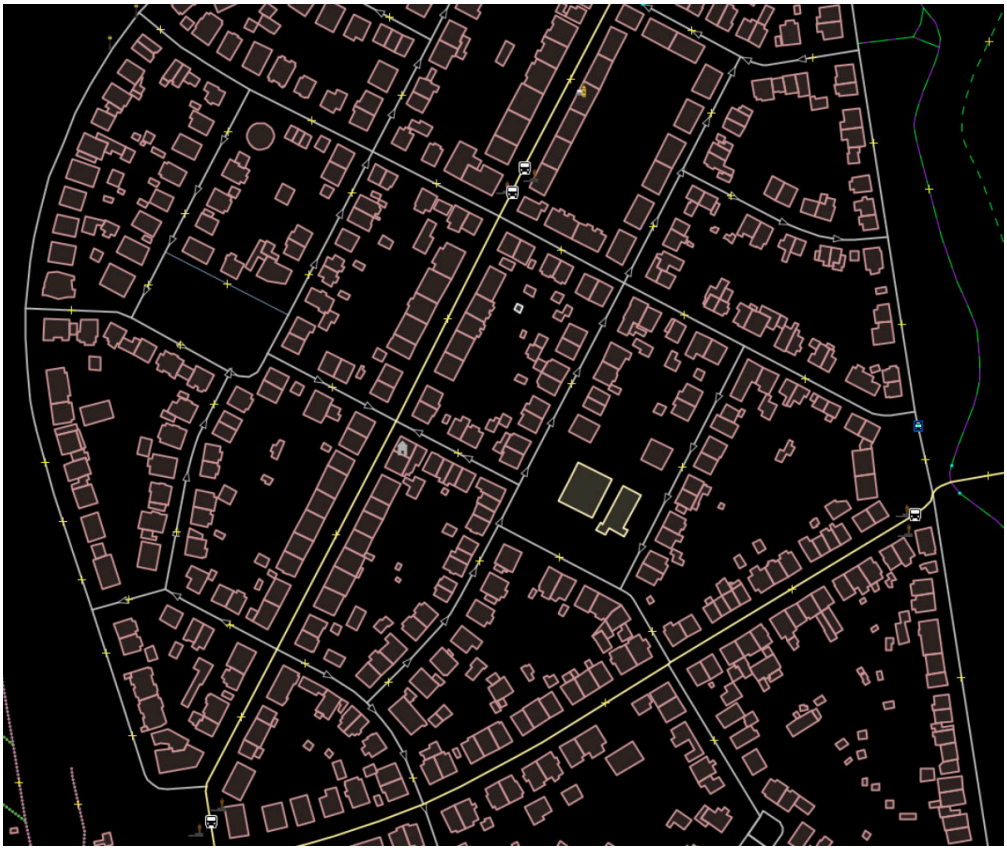


Figure 2.5: An example of a topological map from OpenStreetMap with road network and buildings' outlines.

building footprint, railway and bus stop locations.

A recent trend in the research community is to exploit topological maps to localize a mobile autonomous robot. In [59], the authors propose to match measurements from an onboard LiDAR sensor with buildings' outlines gathered from OSM to localize a vehicle. A similar approach [60] use buildings' footprints to constrain the feasible poses of the robot in a SLAM pipeline. Information from OSM has also been combined with visual odometry to perform global localization within a city [61] or for improving robot localization [62]. Finally, the authors in [63] propose to detect the road surface in point clouds from an onboard LiDAR sensor and match it with OSM road data.

2.3 Artificial Neural Networks

In the last decade, the interest in artificial neural networks has increased drastically, from both academia and industry. The origin of artificial neural networks dates back to 1943 when the first mathematical model of a neural network was formulated to emulate the human thinking process [64]. This model is known as McCulloch–Pitts neuron. Later, in the 1950s, different attempts to emulate a neural network were made and, even if the first attempt failed, in 1959 a working artificial neural network was proposed [65] and effectively applied to a real-world problem, *i.e.,* it removes echoes in phone calls.

Modern neural networks are based upon the Perceptron model [66] proposed in 1957. The Perceptron is a generalization of the McCulloch-Pitts neuron and, differently from the latter, it has learnable weights and real valued inputs. A Perceptron model takes n inputs $x_i \in \mathbb{R}$ that are first multiplied by a set of learnable weights w_i and then summed together. A bias b is also added to the summation, and the resulting value is processed by a non-linear "activation function". An example of a Perceptron with a "step" activation function is depicted in Figure 2.6. Although the Perceptron is limited in its complexity (in 1969 was proven that it could not model the exclusive-or circuit [67]), multiple Perceptrons can be stacked in order to solve more complex tasks (*e.g.,* multi-class classification, non-linear function approximation). In this case, the model is called Multilayer Perceptron (MLP), and an example is depicted in Figure 2.7. However, in 1969 the authors of [67] discovered that the learning rule proposed in [66] was not applicable to multiple layers of Perceptrons, slowing the neural network research progress for a decade.

The learning algorithm typically used to train modern neural networks is called Back-Propagation (BP). Even though different researchers have investigated some concepts of BP in the 1960s, the first efficient version of the BP algorithm as we know it today was proposed in 1970 [68], and the first application of BP for multilayer Perceptrons was introduced only in 1982. The possibility to learn the weights of multilayer Perceptrons, together with the mathematical proof that they can approximate any continuous function [69], have increased the researcher's interest in artificial neural networks again.

During the 1990s, artificial neural networks were successfully applied in many fields, such as image classification [70], autonomous driving [71], reinforcement learning [72] and recurrent neural networks [73]. However, the high computation required by neural networks to be effective has hampered their diffusion in favor of Support Vector Machines

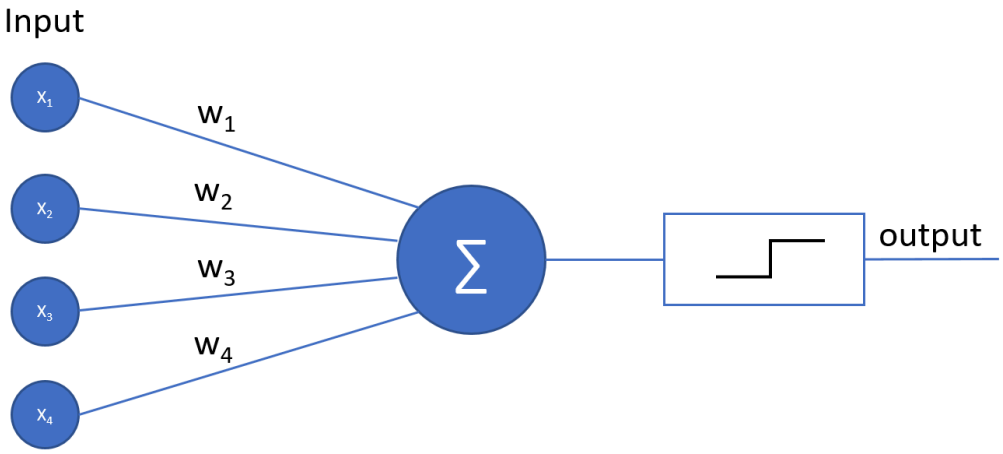


Figure 2.6: The Perceptron model for binary classification. The inputs x_i are multiplied by the weights w_i and then summed. A step function decide whether the inputs represent a positive or a negative sample.

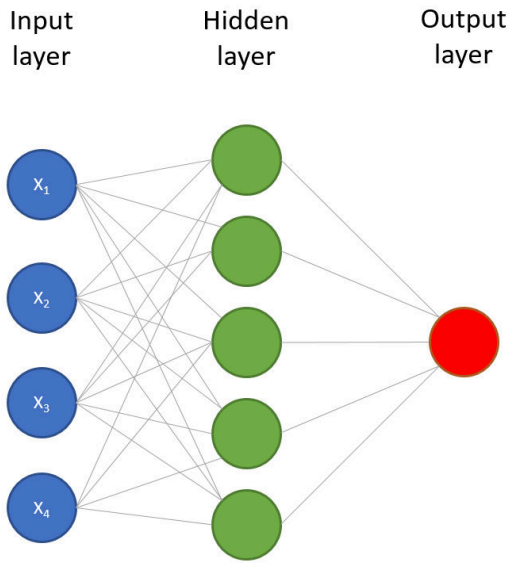


Figure 2.7: An example of a Multilayer Perceptron.

(SVMs) with the "kernel trick" [74]. A SVM is a linear model for classification and regression, in which n-dimensional inputs are separated in two classes by finding the $(n-1)$-dimensional hyperplane that maximize the distance from it to the nearest input on each side. The kernel trick allows SVMs to perform non-linear classification and

regression by mapping the input data into a higher dimensional space where the data are linearly separable. One of the main advantages of the kernel trick is that there is no need to compute the mapping of the input data, but only the inner product of pairs of points in the higher dimensional space is needed to be computed. With the computational power available at that time, SVMs have been proven to achieve comparable classification accuracy compared to neural networks [75], while being faster to train.

Only during the late 2000s the highly parallel computational power available and a new algorithm for learning the weights [76] made the training of deep neural networks possible.

2.3.1 Convolutional Neural Networks

Classical MLPs consist of multiple layers of Perceptrons, and each neuron in a layer is connected to all the neurons in the previous layer. Due to the fully connected nature of MLPs, the weights to be learned increase quadratically with the number of neurons in each layer. Therefore, when the size of the input is huge (*e.g.,* an image), it is impractical to build a deep MLP.

In 1980, inspired by the flow of information in a cat's visual cortex, Kunihiko Fukushima proposed the Neocognitron model [77], an artificial neural network designed for pattern recognition in images. The Neocognitron was composed of S-cells that respond to local features and C-cells, designed to detect more complex features. For example, an S-cell may detect an edge in a specific position, while a C-cell activate if the same edge is present anywhere in the image, *i.e.,* C-cells are spatial invariant. A few years later, in 1989, the first modern CNN [78], called LeNet-5, was proposed. It combined the key ideas of the Neocognitron with the BP learning algorithm in order to classify handwritten digits.

The key innovations of CNNs with respect to classical MLPs are the following:

- **Local Receptive Field** each neuron in a layer is connected only to a set of spatial neighbor neurons in the previous layer. In this way, a neuron can detect simple spatial feature, *e.g.,* edges or corners.

- **Weight Sharing** following the idea that if a feature is relevant in a specific location of the image, the same feature is probably also relevant in the whole image, neurons in a layer are organized in "planes". Inside a plane, all the output neurons are forced

to have the same set of weights, *i.e.,* the same convolutional kernel is applied at every location of the input. Each layer of a CNN is composed of multiple planes, and the output of each plane is called "feature map".

- **Subsampling** after each convolutional layer, a sub-sampling operation is applied to all the feature maps, in order to decrease the spatial resolution by half.

The combination of convolutions and subsampling layers has a twofold purpose: on the one hand, it allows the detection of higher-level features and on the other hand, it reduces the sensitivity of the output to spatial shift and distortion. Moreover, the weight sharing technique drastically reduces the number of free parameters of the network. For example, in the first convolutional layer of LeNet-5 the number of free parameters is ca. 700 times less than the number of connections, and ca. 32 000 times less than a MLP with the same number of neurons. Figure 2.8 depicts the architecture of LeNet-5.

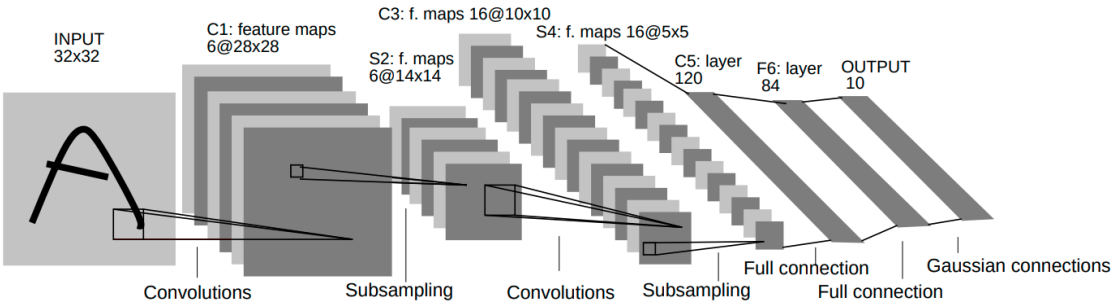


Figure 2.8: The architecture of a typical CNN for image classification, where convolutional and subsampling layers are alternated and then followed by one or more fully connected layers. Image taken from [78].

Even though LeNet-5 achieved state-of-the-art accuracies for handwritten digits classification, the high computational resources and the time requirements for training purposes, led the research community to prefer SVMs over CNNs for more than 20 years.

CNNs gained massive interest from the research community in 2012 thanks to the high parallel computational capacity of Graphics Processing Units (GPUs) and the huge amount of training data available. AlexNet [79] was the first CNN-based approach to win the ILSVRC (ImageNet Large Scale Visual Recognition Challenge) [80] outperforming all the other methods by a large margin. One of the key aspects of AlexNet was the use of the Rectified Linear Unit (ReLU) activation function in convolutional layers. Although ReLU is not completely differentiable, it has been proven that networks with ReLUs train

several times faster than networks with sigmoid or tanh activation functions [79, 81].

Thanks to the impressive results of AlexNet, the interest and efforts of the research community in CNNs has grown from year to year, drastically increasing their performances. One of the most significant achievements was reached in 2015 when, for the first time, a CNN-based approach has beaten human performances on image classification [3]. During the last decade, CNNs have been successfully applied in many fields, outperforming classical machine learning approaches, *e.g.,* medical image segmentation [4], natural language processing [5], playing games [6] or videogames [82], and many more.

2.3.2 Recurrent Neural Networks

While CNNs were specifically designed to process spatial data (such as images or point clouds), another class of ANNs, namely Recurrent Neural Networks (RNNs), were designed to handle data sequences (such as texts, genomes or time series). The first early model of a RNN was proposed in 1982 by John Hopfield [83]. This model, namely Hopfield network, tried to mimic the associative memory of the human brain, and it was designed to learn a set of patterns and recognize them even with incomplete or corrupted inputs. The basic idea behind RNNs is that the output for a specific input does not depend only on the input itself, but also on the previous inputs. The learning algorithm commonly used to train RNNs is called Backpropagation Through Time (BPTT), and is an extension of the previously presented BP.

Modern RNNs are based on the Long Short-Term Memory (LSTM) unit, invented by Sepp Hochreiter and Jürgen Schmidhuber in 1997 [73]. An LSTM is usually composed of four main components: a *cell state*, an *input gate*, a *forget gate* and an *output gate*. The cell state is the *memory* of the unit, and can carry information through time. The input gate and the forget gate control how the cell state should be updated. In particular, the forget gate decides how much of the previous cell state should be kept, while the input gate controls whether to add new information or not. Finally, the output gate controls the actual output of the LSTM layer, by combining input and cell state.

LSTMs have been successfully applied in many tasks that involve the processing of data sequences, such as speech recognition [84], machine translation [85], image captioning [86] and image generation [87]. Major technology companies have implemented LSTM-based techniques in their products: Google Translate, Apple Siri, Amazon Alexa, Microsoft Cortana.

2.3.3 DNNs for Autonomous Driving

In the autonomous driving field, deep neural networks have been exploited to solve numerous tasks, such as detecting surrounding objects (*e.g.,* pedestrians, other cars, road signs) [4, 7–12, 88], geometric reconstruction [13–16], optical flow estimation [17–20], visual odometry [89–91] and vehicle localization [21–23, 92–94]. In this section, we will briefly describe each of these tasks, together with the corresponding most relevant approaches.

The problem of detecting the objects surrounding the vehicle is probably the most critical task for safe navigation. Being able to avoid collision with other cars, stop if pedestrians are crossing the road and detect obstacles are some of the critical capabilities that an autonomous vehicle needs to perform when driving in dynamic environments. This problem can be tackled by exploiting textural information from a camera sensor, and it can be subdivided into three complementary tasks: Object Detection, Semantic Segmentation and Instance Segmentation. On the one hand, in object detection, the goal is to draw a bounding box around each interesting object present in the image and classify the bounding boxes with the respective object class. This allows the identification of all the objects in the scene, while distinguishing different instances of the same class, but it does not provide a so called fine-grained classification. On the other hand, in the semantic segmentation task, we want to classify each pixel of the image with the corresponding class. However, even with this more detailed information, we still cannot distinguish between different object instances, *e.g.,* we are unable to know if two pixels classified as car belong to the same car or not. The instance segmentation is the mix of the previous two tasks: each pixel is classified with the respective class and instance. Figure 2.9 depicts an example of the three tasks mentioned above.

Object Detection and Segmentation

One of the most significant CNN-based approach to deal with the object detection task was R-CNN [95], in which a region proposal step is employed to generate around 2k bounding boxes. Each of these boxes is then fed to a CNN-based feature extractor and is finally classified with a SVM. R-CNN outperformed all the previous approaches by a large margin, but the numerous forward passes of the CNN make it very slow. To speed up the inference time of R-CNN, an extension, named Fast R-CNN [96], that only needs a single forward pass to extract the features, was proposed. In a further extension, Faster

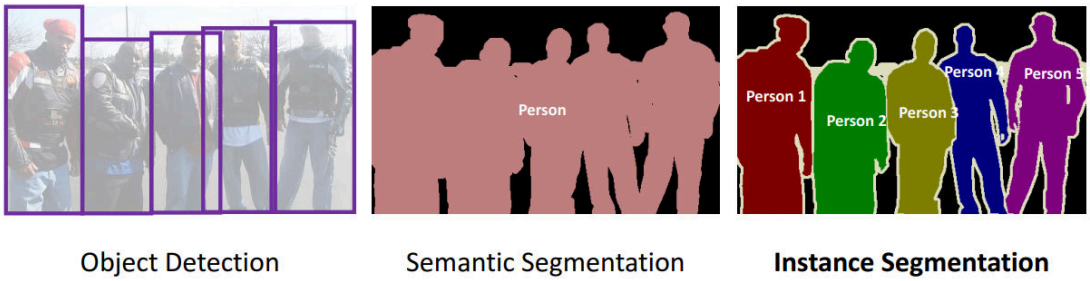


Figure 2.9: Three complementary computer vision tasks: Object Detection, Semantic Segmentation and Instance Segmentation.

R-CNN [7], the authors proposed the Region Proposal Network (RPN) that takes as input the features extracted from the image, and outputs a set of bounding boxes, avoiding the need for a separate prior region proposal step, and reducing the computational time. The improvements proposed in Fast R-CNN and Faster R-CNN drastically reduced the processing time, but not enough to make them suitable for real-time applications.

The latter approaches [7, 95, 96] follows a two-step classification pipeline (first multiple regions are proposed, then each of them is classified). Other approaches that only need a single step also exist, where both bounding boxes (with associated confidence) and classes are predicted at the same time, and the most significant approaches of this type are YOLO [8, 97] and SSD [9]. Single-step detectors are usually faster (the tiny version of YOLO can run up to 220 frames per second) but have lower precision than region proposal based detectors.

In 2015, the first CNN-based approach for semantic segmentation, named Fully Convolutional Network (FCN) [10], was introduced. The authors proposed to replace the fully connected layers of a standard classification CNN (*e.g.,* AlexNet) with convolutional layers. Therefore, the output of the encoder is a coarse feature map instead of a single probability distribution over the classes. Moreover, this made it possible for the network to accept images of arbitrary shape. Since the output feature map of FCN has a low resolution (1/32nd of the input image), to predict a finer-grained output the authors proposed to combine the coarse map with information from shallow and fine layers. A similar approach, named U-Net [4], was proposed to segment medical images. It consists of an encoder that extracts features from the image, and a decoder that classifies each pixel by combining information from different scales using skip-connections. Figure 2.10

depicts the architecture of U-Net. Many recent CNN-based semantic segmentation approaches follow the same idea of U-Net [11, 12, 98, 99].

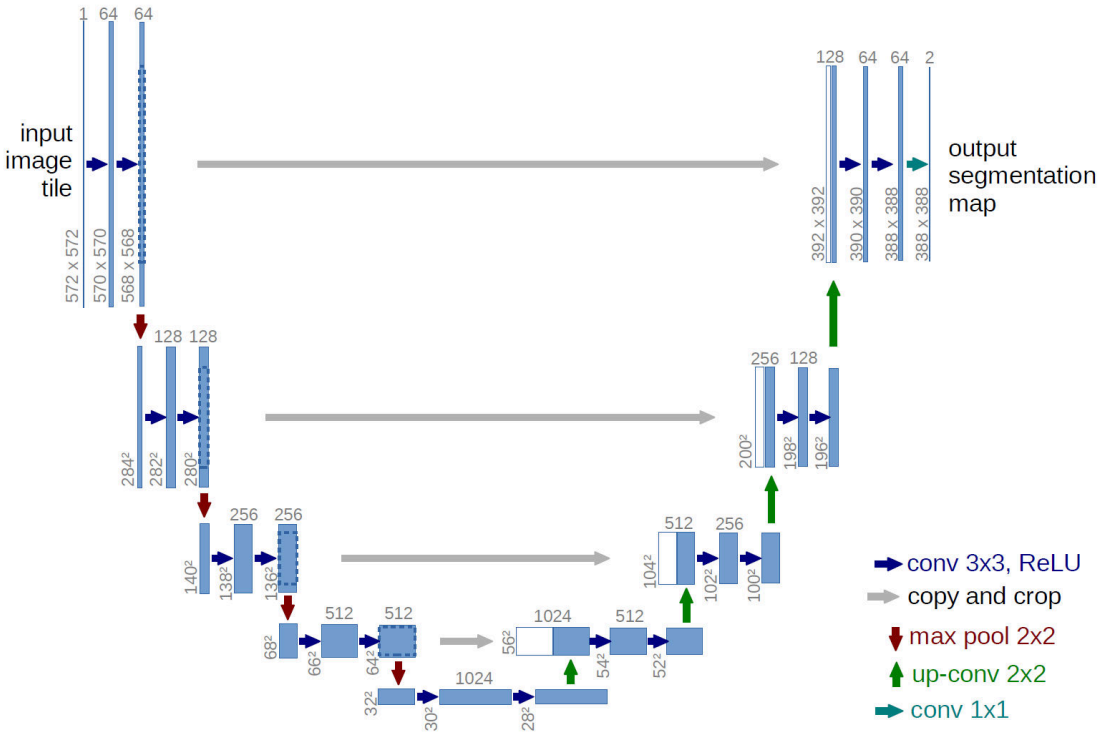


Figure 2.10: The architecture of U-Net: encoder-decoder with skip-connection. Image from [4]

The last task for detecting the surrounding objects (instance segmentation) is a mix between object detection and semantic segmentation. In fact, the most relevant work, named Mask R-CNN [88], combine an object detection network (Faster R-CNN) with an additional pixel-level mask prediction branch.

3D Object Detection

Thanks to the impressive results achieved by CNNs in computer vision tasks, deep learning approaches have also been applied in autonomous driving related tasks that do not directly involve camera images. For example, DNNs have been applied to 3D object detection in LiDAR data, by exploiting birds-eye view elevation images [100, 101], by employing neural networks for point cloud processing [102–104], or by combining LiDAR and camera information [105–107].

DNNs for 3D LiDAR data processing is a new and emerging field. The first DNN-based approaches for point clouds processing [108, 109] first converted the cloud into a 3D occupancy grid, and then applied 3D CNNs. More recently, neural networks that directly represent the cloud as a list of points have been proposed [110–112]. Hierarchical data structures (such as octrees and kd-trees) have also been exploited within DNN approaches [113, 114]. A more in-depth review of existing DNN-based methods for point cloud processing is reported in Section 7.1.

Geometric Reconstruction

Convolutional Neural Networks have been successfully applied also to geometric reconstruction related tasks. According to the KITTI benchmark leaderboard [115], in the last years CNN-based approaches have outperformed standard methods in stereo disparity estimation [13–16], monocular depth estimation [116–118] and optical flow [17–20]. Some attempts to face the visual odometry task with CNNs have also been carried out [89–91], but they have not reached the performance achieved by standard approaches.

Pose Regression

One recent trend in the research community is to exploit CNNs to solve the pose regression task, *i.e.,* predict the pose of the observer given only a single image. The first approach to tackle this problem using a CNN was PoseNet [21], where a deep convolutional network was trained to directly regress the 6-DoF pose of the camera given a single RGB image.

Starting from this work, additional improvements have been proposed by introducing new geometric loss function [92], by exploiting the uncertainty estimation of Bayesian CNNs [93], by including a data augmentation scheme based on synthetic depth information [94], by using the relative pose between two observations [22], or by exploiting multi-learning task [23]. All the latter methods are trained in the working area, and thus they cannot perform localization in environments that have not been seen during the training phase, *i.e.,* they need to be retrained in each area where the network is deployed.

The focus of this thesis is not primarily aimed at proposing new DNN architectures for specific computer vision tasks. Instead, our focus is to exploit DNNs for improving vehicles' localization. In the first part of this thesis (Chapters 4 and 5), we integrate

CNN-based approaches for semantic segmentation and stereo disparity estimation within a localization pipeline. The goal is to localize a vehicle in topological maps by matching high-level features (road intersections and buildings) detected using cameras with their counterpart in the map. In the second part, instead, we directly use DNNs for estimating the pose of the camera. In particular, we propose two DNN-based approaches for camera localization within LiDAR-maps. In Chapter 6 we tackle the local localization task, *i.e.,* the goal is to refine a rough position estimate (for example, gathered from a GNSS), while in Chapter 7 we propose a global localization approach, where we localize the camera without any prior information about its position.

Chapter 3

Ego-lane estimation in highway-like scenarios

In this thesis, we address the problem of vehicle's localization in complex environments with only cheap onboard camera sensors. The first approach, described in this chapter, tackles the problem of localization in highway-like scenarios. Such scenarios are easier than, for example, urban environments, because only vehicles are allowed to transit (*i.e.,* no pedestrians and bicycles), and road markings are well maintained. Accurate metric localization is not required to navigate in highway-like environments autonomously. A combination of adaptive cruise control and lane-keeping features can be exploited to drive inside a lane autonomously, and in the last few years, such features are becoming available on the market (*e.g.,* Tesla Motors' AutoPilot, General Motors' SuperCruise, Audi AI). However, in those systems, tasks such as taking exit ramps or positioning in the correct lane at bifurcations require manual intervention. In order to make the latter tasks autonomous, the vehicle requires a localization with lane-level accuracy. However, lane-level localization is actually a term with broad meaning and it usually might refer to two different problems: on the one hand it might refer to the determination of the lane currently occupied by the vehicle, a problem which is also known as *host-lane* or *ego-lane* estimation; on the other hand, it might refer to the estimation of the lateral position of the vehicle inside the lane or whole road. The latter problem is relevant for the lower-level control of the vehicle, while the first is relevant in the higher-level (tactical) control of the vehicle, *i.e.,* trajectory and maneuver planning. Solutions to the latter live in $\mathbb{R}$, while solutions to the first live in $\mathbb{N}$. In both cases, images obtained from forward-facing camera(s) of the vehicle are usually processed to detect road lines. Those lines are then exploited to infer the vehicle's ego-lane at the time of the image(s)

capture.

In this chapter, we propose an approach to estimate the ego-lane of the vehicle, *i.e.,* in which lane the car is driving. Since different methods to detect road lines have been proposed by the research community [119] since the 1980s [25], instead of developing a new line detector, we propose to exploit existing line detectors coupled with a probabilistic approach aimed to estimate the ego-lane of the vehicle. The method is based on a HMM with a transient failure model, which allows us to accommodate inaccurate or missing road line detections. Differently from other works available in the literature, ours is a modular, hence reusable, algorithm for improving the ego-lane estimation that could be obtained from a generic line detector. The algorithm exploits a GNSS measure only to initialize the model with the correct number of lanes retrieved from the OSM (or other cartographic services) road property "lane number". Being the proposed model independent of the line detector, it is ready to be used over any approach for line recognition, increasing the overall performance of the basic low-level feature detector.

Unfortunately, most of the available dataset for autonomous driving related research either contains highways with few lanes or the recording vehicle stay in the same lane for the majority of the dataset; thus, we were unable to find a suitable dataset for a proper evaluation of the proposed approach. Therefore, we recorded a new dataset, specifically aimed at evaluating ego-lane estimation algorithms, composed of more than 100Km of highway driving in Italy and Spain, and we manually annotated the Ground Truth about the vehicle's ego-lane. This dataset was recorded in wide highways (three and four lanes), and include more than 100 lane changes. We evaluated the effectiveness of the proposed algorithm by employing different line detectors and showing we could achieve much more usable, *i.e.,* stable and reliable, ego-lane estimates compared to relying only on the underlying line detector's output.

3.1 Related work

Ego-lane estimation for autonomous driving has been extensively investigated in the last decades. The first achievements were obtained by the group of Prof. Dickmanns [25]; they introduced a road representation model based on clothoids, which were then updated with the image measurements by using Kalman filters.

Basing on these results, active research has been conducted in subsequent years [26, 120–122]. Heterogeneous modelling techniques for the lane markings (*e.g.,* parabolas,

clothoids, poly-lines or b-splines) were proposed. Typically, road markings are extracted from images after some preprocessing steps designed to remove clutter and irrelevant areas.

One of the most challenging tasks that are to be solved to identify the ego-lane is the detection of the road surface. Achieving good discrimination of the road surface from other parts is crucial since it is the basis for further processing. However, this detection is usually adversely affected by the large amount of clutter normally found on real roads. While faded road markings, unusual or specific weather conditions, or even light variations might severely affect the road surface detection, the visibility of the road surface is quite frequently hampered by the presence of other vehicles, thus requiring different considerations to solve the problem.

Most of the current Advanced Driver Assistance Systems (ADAS) available in modern cars, like the systems presented in Section 2.1 (*e.g.,* Lane Departure Warning (LDW), Adaptive Cruise Control (ACC) or lane-keeping), require just a partial understanding of the whole observed scene, such as the lines of the vehicle's lane or the lane crossing points [119, 123].

For what concerns the sensors, even though LIDAR-based algorithms sport the advantage of active lighting, vision-based algorithms are, as for today, the most frequently used sensing techniques for line detection and ego-lane estimation for two main reasons. First, camera sensors are likely to be found in most of the modern cars available, and second, road markings are designed to be human-visible in mostly all driving conditions and are usually well maintained in highways. For an excellent review of standard approaches in lane localization, see [119].

With the objective to pursue lane-level localization, the authors in [124] propose to exploit the objects present in the surrounding of the vehicle and to describe the probabilistic dependencies between the object measurements, by means of a factor graph model. A similar proposal comes from the authors of [125], where Histograms of Oriented Gradients are used to align the images acquired from a front facing camera to the road lane markings available in the map, to improve the vehicle's localization.

In order to increase the performance of ego-lane estimation algorithms, many authors propose to exploit additional road information gathered by map services as well as information provided by GNSS. In this regard, an interesting approach is presented in [126], where the authors tackled the ego-lane estimation as a scene-classification prob-

lem. They holistically infer the lane number, leveraging both spatial information and objects around the vehicle, and finally training the best classifier with different learning algorithms. In [127], the author presented a robust lane-detection-and-tracking algorithm combining a particle filtering technique for lane tracking and RANSAC for the detection of lane boundaries. The work detects left and right lane boundaries separately, without exploiting fixed-width lane models, and combining lane detection and tracking within a common probabilistic framework.

The authors in [128, 129], respectively in highway and urban scenarios, propose to exploit boosting classifiers and particle filtering approaches. A similar technique was proposed in [130], where multiple evidence from a visual processing pipeline was combined within a Bayesian Network approach.

Closer to our proposal are the works in [131–133], where the authors explicitly address the multiple-lane detection problem. In [131] multiple lane detections are performed after a first processing phase, where the authors identify the ego-lane geometry. Then, adjacent lanes are first hypothesized and then tested, assuming same curvature and width for all lanes, a fair assumption for most of the multi-lane roads, including highways. Similarly, the work proposed in [132] also considers highway scenarios and parallel lane markings, with respect to the detected ego-lane. The authors in [133] proposed a multi-lane detection algorithm also based on a hypothesis generation and testing scheme, ensuring an accurate geometric estimation using a robust line fitting pipeline and vanishing point estimation.

More recently, end-to-end CNN-based approaches for multi-line detection have been proposed to avoid the designing of hand-crafted features. The task of lines detection with neural networks can be treated as a semantic segmentation problem (Section 2.3.3), *i.e.,* classify each pixel as belonging to a line or not [134]. Alternatively, instance segmentation approaches can be exploited to distinguish pixels belonging to different lines [135]. Several techniques have been proposed to improve CNN-based lines detector. In [136] a multi-task CNN guided by the vanishing point (defined as "the nearest point on the horizon where lanes converge and disappear") detect and classify lane and road markings. The authors of [137] employed a Generative Adversarial Network (GAN) to produce more *realistic* and *structure-preserving* segmentation outputs. A message passing scheme that enables spatial communication between pixels across rows and columns in a layer has been proposed in [138]. Finally, the most recent work presented in [139] employed

lightweight pre-trained CNNs coupled with a novel *Self Attention Distillation* module, substantially improving lines detections without any additional labels or supervision. Approaches based on CNNs are more robust to visual changes (*e.g.,* light or weather changes) than hand-crafted based methods, achieving reasonable performance in night and rainy scenarios.

Differently from the aforementioned contributions, where the authors proposed new detection pipelines for the ego-lane estimation problem, in this chapter, we introduce a generic scheme aimed at improving the ego-lane estimation capabilities of potentially every line detector. Additionally, the output of a lane detection algorithm could be fed into our algorithm to increase its performance in ego-lane estimation.

3.2 Proposed algorithm

The goal of the proposed algorithm is to estimate the vehicle ego-lane in highway-like scenarios, when the topology of the roadway does not change, *e.g.,* because of exit ramps or bifurcations. The input of the proposed approach are both a global localization, at the level of accuracy provided by today GNSS, and the detections of the road line markings.

Our proposed method is designed to tolerate occasional temporary failures of the underlying line detector as well as its noisy measurements. A line detector is a software component that detects and tracks the relative position of both dashed and continuous road lines, with respect to the vehicle.

Our algorithm relies on a probabilistic model designed to be independent of the line detector, so we can compare the results of our algorithm when working on the output of different line detectors. Indeed, the estimation of the vehicle's ego-lane can be regarded as a consequence of the outcome of the line detection procedure. The position of all the road lines with respect to the vehicle allows determining the ego-lane by using simple geometric considerations, on a per-frame basis. Unfortunately, line detections are usually not fully reliable, being hampered by faded road markings, cluttering elements from the nearby traffic, or weather conditions, see *e.g.,* Figure 3.1, 3.2a and 3.5.

However, we combine the line detections with an index about how reliable each detection is, we call this index *Line Reliability Index* (LRI). This index, together with our proposed probabilistic model, allows us to handle the noisy output of the line detectors properly.

Furthermore, consider the situation depicted in Figure 3.2a, surely a critical situation

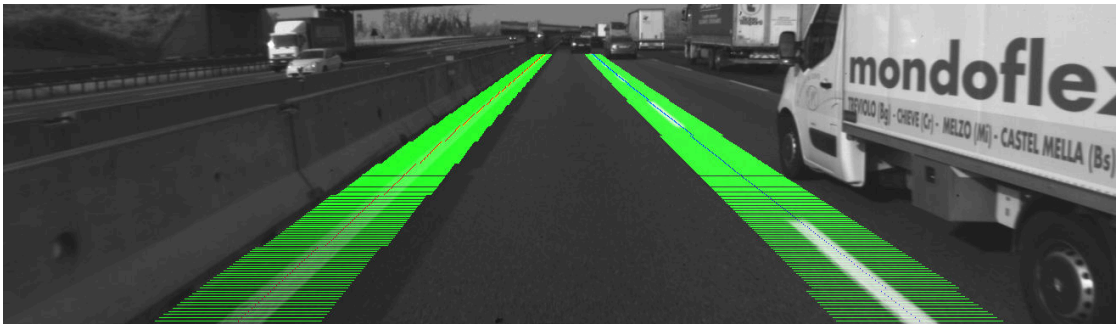


Figure 3.1: An example of a moderately congested condition on the A4 (Turin - Milan - Venice - Trieste) highway, Italy. Even at this moderate level of congestion most road markings are hidden by traffic.

(a) **(b)**

Figure 3.2: In (a) only one line out of four is detected, thus the highlighted lanes in (b) have a higher probability of being the vehicle ego-lane, as implicated by the distances to the detected lines.

for ego-lane estimation, due to the shadow that hampers the line detection. Even though the exact lane cannot be estimated from the only detected line, the distance measured from such line would limit the uncertainty only to the compatible lanes, as depicted by the green highlighted lanes in Figure 3.2b.

Our proposal is to tackle the ego-lane estimation with a probabilistic model, in order to infer the ego-lane by leveraging consecutive, yet incomplete, observations over time. From a technical perspective, we propose a HMM with a *Lane* variable that can take n values, corresponding to the number of lanes retrieved from an OSM-like service, and a *Sensor State* variable that represents the reliability of the underlying lane detector.

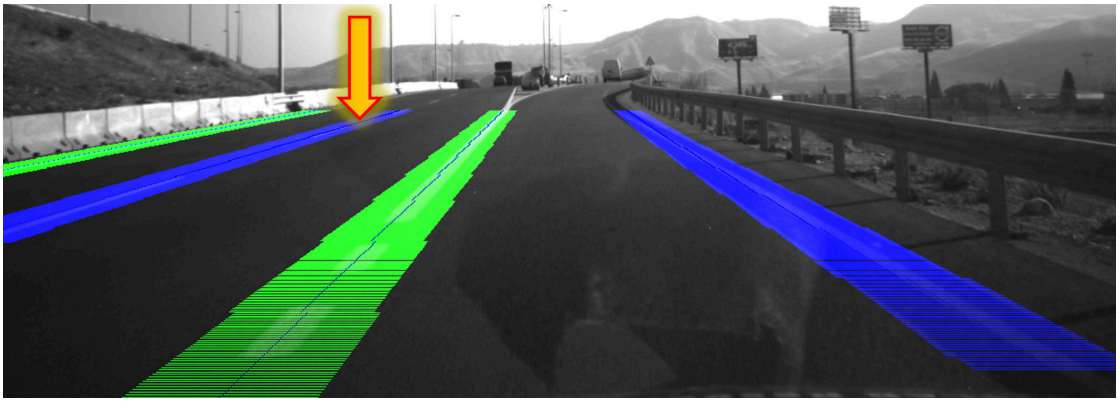


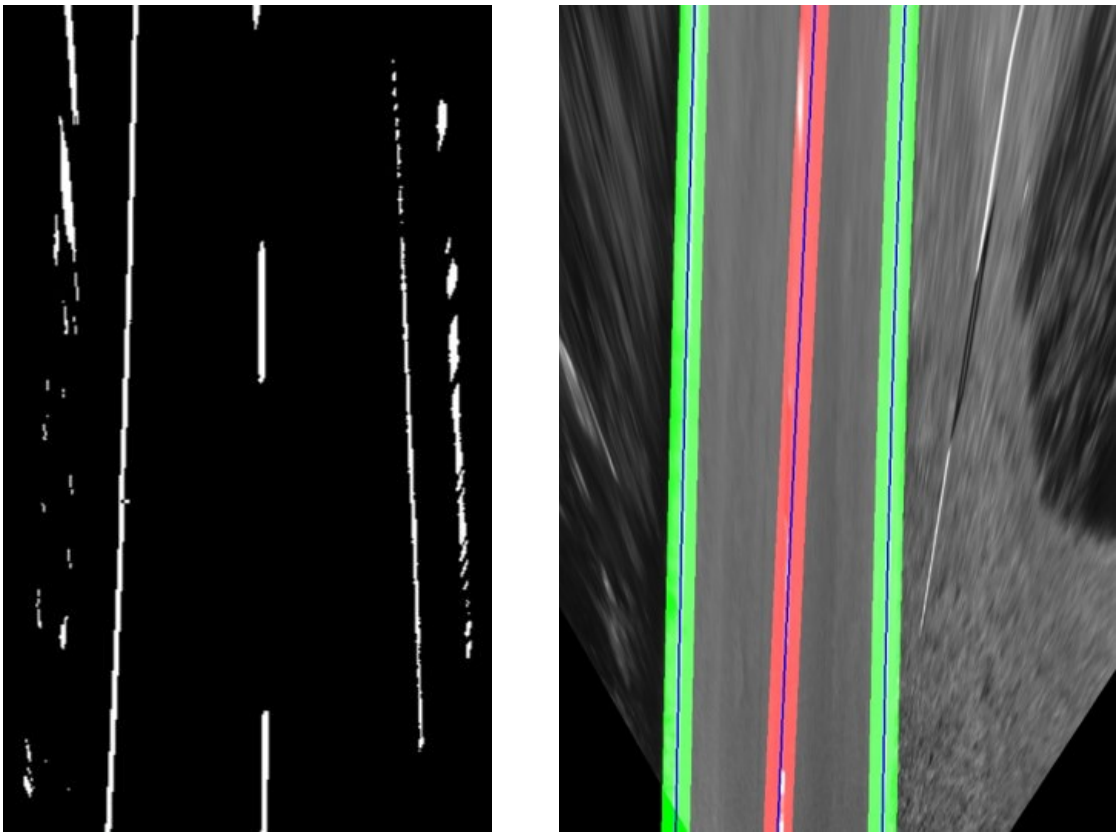
Figure 3.3: If only the line indicated with the arrow were detected, the probability of being in Lane$\{1|2|3\}$ would be $\{0, 0.5, 0.5\}$. The idea of exploiting the plausibility given by each line, repeated for all the detected lines, is combined with our model for solving the ego-lane estimation problem. Here, the green and blue visually suggest the reliability of the lines (green is higher).

3.2.1 Line detection and tracking

In this subsection, we shortly describe the line detection and tracking algorithms used in the experimental activity. We first introduce a "simple" line detector and tracker capable of working with both a stereo and a monocular camera configuration, provided that the projection of the camera(s) is calibrated with respect to the vehicle reference frame before its usage. This algorithm is a modified version of a software kindly provided by the INVETT Research Group of the Universidad de Alcalá, and consists of the following steps:

- The contours of the road markings are extracted from the Bird's-Eye / Inverse Perspective view (BEV / IPV) of the left camera image and discarded if their area is below a threshold (Figure 3.4a). To compute the BEV / IPV image, a homography matrix is computed, basing on the intrinsic values of the projection model, and the extrinsic values with respect to the road surface. The contours in the BEV image are then determined using the algorithm proposed in [140].

- The algorithm then tries to fit, onto the detected contours, a fixed number of lines or clothoids (both the number and the type are parameters), trying to cover the highest number of contour areas (Figure 3.4b); if the stereo configuration is available, the algorithm exploits it to exclude lines / clothoids not lying on the ground plane. The ground plane equation is evaluated using the output of the



(a) Contours extraction **(b)** Lines fitting

Figure 3.4: An example of the simple line detector employed. First, road markings contours are extracted from the BEV image, then multiple lines (or clothoids) are fitted to cover the highest number of contour areas.

SGBM [141] or the ELAS [142] stereo-matching algorithm (also this choice is a parameter). Concluding, both a monocular and a stereo version of the algorithm are available.

- The parameters of each line/clothoid are then updated through a Kalman filter.

An example of this simple line detector is depicted in Figure 3.4.

With respect to the previous k (*e.g.*, $k = 10$) frames, the number of times the same line is detected is taken as the *Line Reliability Index* (LRI). Furthermore, again for each line, the LRI is used to set a flag, named *isValid*, once the counter reaches its maximum value; to reset the flag the counting goes on with hysteresis, so that the flag is not reset until LRI goes below a certain fraction of its maximum value.

Box 3.1: The line detector output for the image in Figure 3.3. The isValid flag is set to TRUE when LRI=10, and reset using a hysteresis counting procedure. A negative offset refers to lines on the left of the vehicle.

Line1: isValid = 1; continuous=1; LRI: 10; offset: $-9.15\,\mathrm{m}$
Line2: isValid = 0; continuous=0; LRI: 09; offset: $-6.47\,\mathrm{m}$
Line3: isValid = 1; continuous=0; LRI: 07; offset: $-2.15\,\mathrm{m}$
Line4: isValid = 0; continuous=1; LRI: 00; offset: $+0.99\,\mathrm{m}$

This simple line detector and tracker achieves good performances only under optimal illumination conditions and, as depicted in Figure 3.3 and shown in the corresponding results in Box 3.1, dashed lines and shadows are not always handled correctly. However, despite its limited performances, it allowed us to evaluate the effectiveness of our contribution, which is designed to enhance the vehicle ego-lane estimation by exploiting a noisy sensor as well as the *road lane* property gathered from an OSM-like service.

Moreover, to effectively assess the proposed probabilistic model, we performed an extensive search to find other multi-line detectors that included a measure of the reliability of the detection of each line. Although the literature about generic line detection is huge, as the problem has been investigated since the beginning of digital image processing, solutions dedicated to detecting multiple road lines that also provide freely available software reduce, to the best of our knowledge and at the time of the investigation, to the following two.

- The proposal by Mohamed Aly [143], which exploits a robust approach based on a line / Bezier line tracker and a RANSAC procedure. As far as their publicly available results show, this algorithm is able to detect all the lines on the road surface, making it one of the most complete and well performing algorithms. Nevertheless, on the one hand, the publicly available software does not include the tracking module, and on the other hand, the number of parameters is overwhelming (about one hundred). These aspects make this software extremely hard to use. We have not been able to find a reasonably good configuration for the datasets used in our experiments. In conclusion, this option, although very appealing, could not be used in our experimental activity.

- The approach by Hur [144] (MLD in the following), which detects a maximum of four lines, corresponding to the markings of the current lane as well as of the two

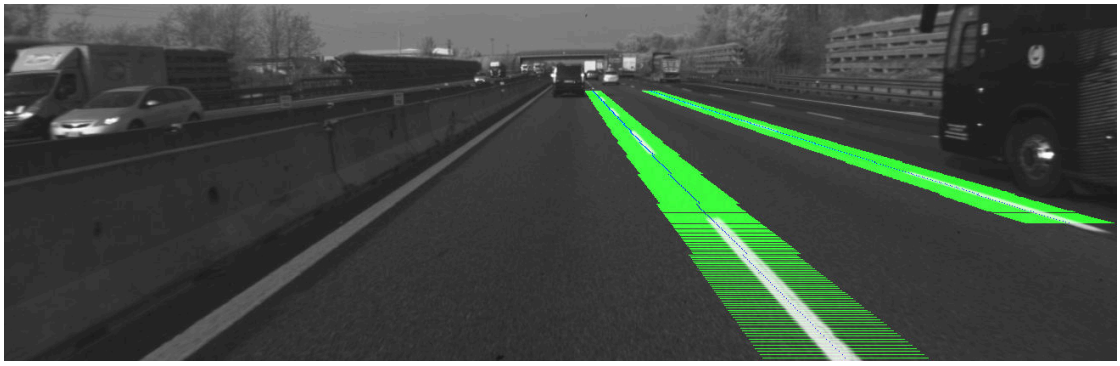


Figure 3.5: In this figure, just two out of the five lines are correctly detected and tracked. The shadow created by the Jersey barrier prevents the correct detection of the leftmost line, although in our opinion the line might be detected, perhaps with a different set of parameters, which might in turn bring in other misdetections. An error might also arise with dashed lines, whenever the space between two consecutive detected dashes is increased by some vehicle. The two rightmost lines are not detected because of their limited thickness; also in this case a different processing pipeline could detect such lines, usually introducing other errors.

neighboring lanes. This software was adapted to our requirements for this work by introducing a procedure for determining whether a line is dashed or continuous.

Neither of the two solutions can exploit a stereo camera configuration, and both rely on the extrinsic projection parameters to determine the distance of each line from the vehicle, exploiting the BEV / IPM image.

3.2.2 Tentative vector and reliability of the whole detection

To exploit the measurements provided by a line detector and tracker like the ones mentioned above, we derived a probabilistic (inverse) sensor model, which exploits both the spatial information carried by the lines and the LRIs, both produced by the line detector and tracker. The processing pipeline is therefore composed as follows. First, the lines are sorted, in ascending order, based on their lateral offset with respect to the vehicle. Then, a vector of counters, of the size of the number of lanes, is created. This vector is called *tentative* (implied: *distribution of the belief on the state as from the measurements*, as usual for an inverse sensor model). The values in this vector are determined by iterating the following steps for all the valid lines, *i.e.,* the lines whose *isValid* flag is set, taking into consideration whether each line is dashed or continuous.

- One is added to the i-th tentative vector value if it is in accordance with the

measurement, *i.e.,* if being in the i-th lane is compatible with the line position; this has the objective to cumulate the plausibility of being in the i-th lane, given the detected lines.

- If the line has the continuous flag set, an additional *Bonus Value* (BV) is added to the tentative vector position (based on the distance with respect to the line); this has the objective to represent the fact that continuous lines are more informative, as they are usually the leftmost/rightmost lines of the road.

For example, after the evaluation of the line indicated with an arrow in Figure 3.3, the resulting tentative vector would be $[0; 1; 1]$.

During the iteration on all the lines, we also accumulate all the LRI counters and compute the fraction over the maximum LRI value times the current number of expected lines. This value is taken as an index of the overall reliability of the detector, namely a Whole Output Reliability (WOR). This, in turn, can be taken as an observation of the sensor being properly functioning or not.

It has to be noticed that some of these rules could be not adequate for all line detectors. For example, if a line detector could not provide a *continuous* flag or a reliability index for each line, the set of rules must be modified, in order to provide a frame-level tentative vector and an overall WOR.

3.2.3 HMM with Transient Failure Model

As we previously mentioned, typical line detectors could fail to detect all the road lines due to, for example, shadows or traffic occlusion. To tackle this problem, we applied a filtering algorithm based on a HMM, which includes a variable representing the functioning of the sensor itself. For an introduction to HMMs, see [145]. The proposed model allows us to take advantage of incomplete and noisy road line observations in a probabilistic fashion, to better estimate the current ego-lane as well as whether the sensor is properly working or not. In our opinion, this extra degree of freedom, *i.e.,* the explicit modelling of the sensor functioning state, yields better performance, compared to consider the ego-lane as the only information, as it gives an extra area of accommodation for matching the unknown value of the state variables to the observations.

The HMM implements a filtering procedure over discrete random variables, where each iteration depends on the parameterization in Eq. (3.1), see below for an explanation of

the parameters.

$$\text{HMM}(n, \sigma_1, \sigma_2, p_1, p_2, p_3, p_4, BV) \tag{3.1}$$

There are four variables, for each time frame, see Figure 3.6:

- Lane: represents the lane where the vehicle is (believed to be) located. This is a scalar variable, taking discrete values, which identify one of the road lanes.
- Sensor State (also SS in the following): a scalar variable, taking one out of two values, which represents whether the sensor is properly working or not, and the taken values could be OK or BAD.
- Detector Output: as we developed an inverse sensor model, this variable is homogeneous to the Lane variable; thus, it is a scalar value in the range $[1 \dots n]$.
- Reliability Index (WOR): analogously to Detector Output, it is homogeneous to the Sensor State variable and can take one value out of the OK, BAD possible values.

In principle, all the above variables would be represented as scalar values; in practice, they will be represented as probability distributions over their state spaces. In the model, Lane and SS are *hidden* variables, while Detector Output and WOR are *observable* variables. Our goal is to infer, at each time frame, the state of the variable Lane (*i.e.,* in which lane the vehicle is driving), given the evidence on the observable variables.

The dependencies between these variables are described using Conditional Probability Tables (CPTs), *i.e.,* tables that describe the probability distribution of a variable (the values the variable can take are in the columns), given the state of its parents in the graph (the values the parents variables can take are in the rows) of the model in Figure 3.6. In other words, the CPTs allow us to forward compute the expectations on the variables, which will be then updated with the observations.

The first two variables (Lane and Sensor State) have temporal dependencies, while the others only depend on variables within the same time frame.

Table 3.1 describes the dynamics of the lane variable, *i.e.,* the probability distribution over the lanes at time $t + 1$, given the vehicle is on the lane i at time t. Thus, given a value of Lane at time t, we expect the Lane variable at time $t + 1$ to follow a normal distribution with the Lane at time t as mean ($\mu = Lane_t$) and σ_1^2 as variance. The variance σ_1^2 controls how likely it is for the vehicle to change lane during the specific

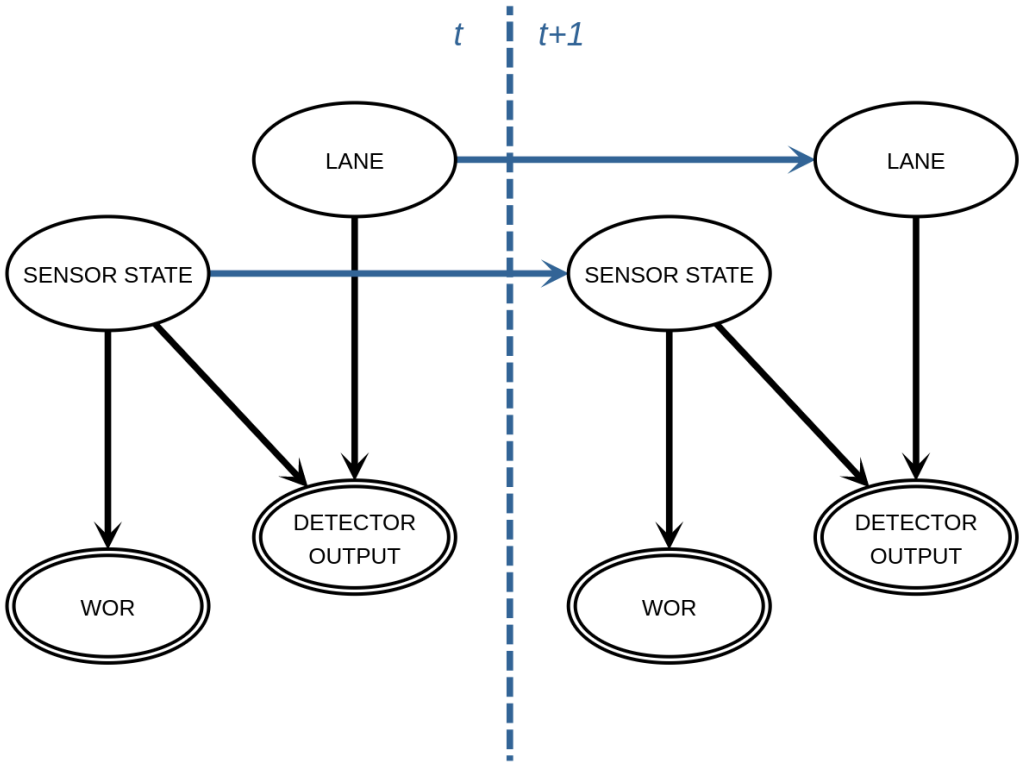


Figure 3.6: Two time frames of the HMM. The single circled variables are hidden, and the double circled variables are observable.

time interval. Notice that σ_1^2 is a parameter of our algorithm, and should be set according to both the actual speed at which the vehicle travels and the frequency of the ego-lane estimation, *i.e.,* the image frame rate. In order to adapt a normal distribution into a probability distribution over the n states that the variable Lane can take, we applied a transformation $\mathcal{F}$ defined as follows. First, for every state $j \in [1 \dots n]$, we assign $p(Lane_{t+1} = j | Lane_t = k)$ to the probability density function of the normal $\mathcal{N}(k, \sigma_1^2)$ in i. Then, we normalize the probability distribution to sum up to 1. For example, the probability that the variable Lane at time $t+1$ is j given it was k at time t is computed as follows:

$$p(Lane_{t+1} = j | Lane_t = k) = \frac{f(j|k, \sigma_1^2)}{\sum_{i=1}^{n} f(i|k, \sigma_1^2)} \tag{3.2}$$

Here, $f(j|k, \sigma_1^2)$ is the probability density function of $\mathcal{N}(\mu = k, \sigma_1^2)$ in j.

Table 3.2 describes the sensor dynamics with respect to its mistakes, which in turn is related to road markings conditions, and lighting conditions. If the sensor is working

Table 3.1: Lane CPT

	Lane_{t+1}			
Lane_t	1	2	...	n
1	$\mathcal{F}(\mathcal{N}(1, \sigma_1^2))$			
2	$\mathcal{F}(\mathcal{N}(2, \sigma_1^2))$			
...	...			
n	$\mathcal{F}(\mathcal{N}(n, \sigma_1^2))$			

Table 3.2: Sensor State CPT

	$\text{Sensor State}_{t+1}$	
Sensor State_t	OK	BAD
OK	p_1	$1 - p_1$
BAD	$1 - p_2$	p_2

properly, it will remain in the properly working state with probability p_1, and switch to giving a wrong output with probability $(1 - p_1)$. If the sensor is not in a properly working state, then it will remain in such a state with probability p_2, and switch back to properly working with probability $(1 - p_2)$. This idea aims, by suitably setting p_1 and p_2, to represent the real experience of the sensor working most of the time properly, and then presenting a sequence of frames with the sensor providing a wrong output.

Table 3.3 describes the detector output with respect to the state of its parents in the HMM: Sensor State and Lane. Therefore, the table has as many rows as the combinations of the values that the two conditioning variables can take, while it has as many columns as the values that the conditioned variable can take. As we have an inverse sensor model, the output of the detector is homogeneous to the Lane variable, thus the table has n columns. If Sensor State is OK, we expect the detector to provide a reliable estimation of the ego-lane. Therefore, we expect the Detector Output to follow a normal distribution with mean equal to the value of the variable Lane, and variance σ_2^2. This variance represents the accuracy of the sensor in determining the ego-lane when properly working; this is clearly a different value from the lane-change dynamics of the vehicle σ_1^2. Again, the same transformation $\mathcal{F}$ defined for the Lane CPT is applied to adapt the normal distribution to a probability distribution. However, if Sensor State is BAD, the

Table 3.3: Detector Output CPT

Parents		Detector Output
Sensor State	Lane	1 2 ... n
OK	1	$\mathcal{F}(\mathcal{N}(1, \sigma_2^2))$
	2	$\mathcal{F}(\mathcal{N}(2, \sigma_2^2))$
	...	...
	n	$\mathcal{F}(\mathcal{N}(n, \sigma_2^2))$
BAD	1	$\mathcal{U}(1, n)$
	2	$\mathcal{U}(1, n)$
	...	...
	n	$\mathcal{U}(1, n)$

Table 3.4: WOR CPT

	WOR	
Sensor State	OK	BAD
OK	p_3	$1 - p_3$
BAD	$1 - p_4$	p_4

detector output will be independent of the real lane, thus the output will be uniformly distributed.

We can similarly define the CPT for WOR, the expected overall reliability of the sensor's output, shown in Table 3.4. As in our model the WOR has only one parent, Sensor State, only two rows are reported in the Table. Notice though that a different model, where WOR depends on the Lane variable also, could be considered, which might be more appropriate for certain situations or line detectors. The parameters p_3 and p_4 represent the probability of a correct evaluation of the Sensor State when the Sensor State is, respectively, OK and BAD.

3.2.4 Inference

Perform inference with our model means to compute the most probable value of the variable Lane, given the evidence on the observable variables Detector Output and WOR. This evidence is computed from the line detector and tracker using the tentative vector

and the overall WOR, as described in Section 3.2.2.

To compute the belief on the hidden variables (*i.e.,* Lane and Sensor State) at time $t+1$, we start from the HMM state at time t. Firstly, leveraging Table 3.1 and Table 3.2, we compute the expectation at time $t+1$ on these variables. *Sensor State* is shortened in *SS*.

$$P(\text{Lane}_{t+1} \mid \text{Lane}_t) = P(\text{Lane}_t) \cdot \text{Lane CPT} \tag{3.3}$$

$$P(\text{SS}_{t+1} \mid \text{SS}_t) = P(\text{SS}_t) \cdot \text{SS CPT} \tag{3.4}$$

Then, in order to incorporate the new evidence carried by the Tentative vector and the WOR index, the Bayes formula has to be applied, so obtaining the belief over the hidden state at time $t+1$. There are, in general, two ways of applying the Tentative vector and WOR index as evidence in the inference. The first way is to simply consider the most probable value of the Tentative vector and WOR index as hard evidence for the belief on the state. The other way, instead, is to consider the Tentative vector and the WOR index as virtual evidence [146], *i.e.,* by setting the evidence as a probability distribution over the state space of the observable variables. Since this second way allows a more comprehensive representation of the evidence, we have only considered this approach, also because the additional computation required for the inference is irrelevant, given the small size of the model. The general way to perform inference using virtual evidence is to create a temporary child node for each observable variables and setting its CPT proportionally to the evidence probability distribution, and then set hard evidence on these virtual nodes.

In Figure 3.7 is depicted an overview of the proposed model. The blue box is the core of our model, *i.e.,* the HMM, which is independent of the detector. The red box is the line detector and tracker, on which we tried to impose no constraint, keeping our model as general as possible. Lastly, the green box is the set of rules used to connect the line detector and tracker output to the HMM. This part may need to be changed with the output format of the line detector and tracker.

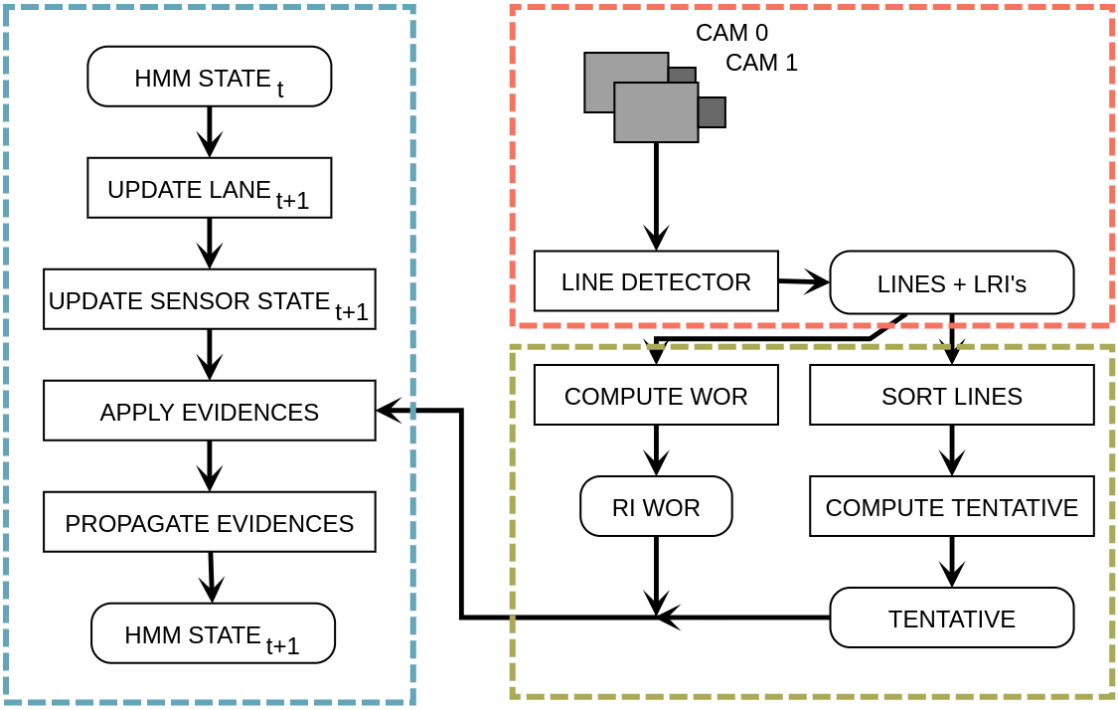


Figure 3.7: An overview of the proposed model.

3.3 Experimental evaluation

To effectively verify the improvements achieved by our model, we collected two datasets in real driving conditions. The first dataset was recorded in the A4 highway, Italy, from Bergamo to Milan. The second dataset is from the A2 highway area of Alcalá de Henares, Spain. Both the datasets were recorded at 10 fps and with a resolution of 1312x540 and 1392x400 pixels respectively. Differently from standard datasets like KITTI, in which the highway sequences only contain few lanes, we drove our vehicles on wider highways with 3 and 4 lanes (Spain and Italy respectively), including more than 100 lane changes in the A4 highway sequences. We manually annotated the ground truth (GT) about the correct lane for more than 20K frames, considering as $Lane = 1$ the leftmost lane as in 3.2a. For each frame, we also included a "crossing flag" to indicate whether the vehicle is changing lane, so to exclude ambiguous lane assignments, see Figure 3.8.

For each experimental setting, the parameterization of Eq. (3.1) was empirically defined after an optimization phase, aimed at identifying the best parameter set with respect to the GT. We tested our approach using four different variants of the "simple"

Figure 3.8: Two frames from the proposed annotated dataset, the detector output is overlaid. In the top image, the vehicle was traveling in the A4 highway, Italy, and performing a lane change, *i.e.,* in the dataset the crossing flag is set. The bottom image depicts a frame from the A-2 highway, Spain.

line detector presented in Section 3.2.1 by changing the camera setup (monocular or stereo) and the geometry of the detection (line or clothoid). Moreover, we tested our approach using the MLD line detector [144]. The parameter values used during the experiments in Italy and Spain are reported in Tables 3.5 and 3.6.

As further research is required for this problem, and to allow future researchers to compare their work with ours, we published our datasets and the associated GT values online[1].

We evaluated the localization performances of our proposal comparing the ego-lane estimates with respect to the GT, for a set of configurations (*e.g.,* with/without the proposed model, different line detectors). The results are presented in Tables 3.7 to 3.26, on a per-frame basis, reporting whether correct lane classifications were achieved.

[1]The dataset and the annotations are available on our lab's website: http://www.ira.disco.unimib.it/ego-lane-estimation-by-modeling-lanes-and-sensor-failures

Table 3.5: Parameterization used for the different experimental settings — Italy (4 lanes).

Run ID	Line Detector	σ_1	σ_2	$p1$	$p2$	$p3$	$p4$	BV
01	ML	0.336	0.696	0.895	0.894	0.690	0.461	7
02	SL	0.481	0.296	0.160	0.970	0.613	0.975	9
03	MC	0.407	0.360	0.853	0.993	0.303	0.640	4
04	SC	0.386	0.598	0.906	0.994	0.311	0.595	7
05	MLD	0.324	0.707	0.223	0.963	0.779	0.873	1

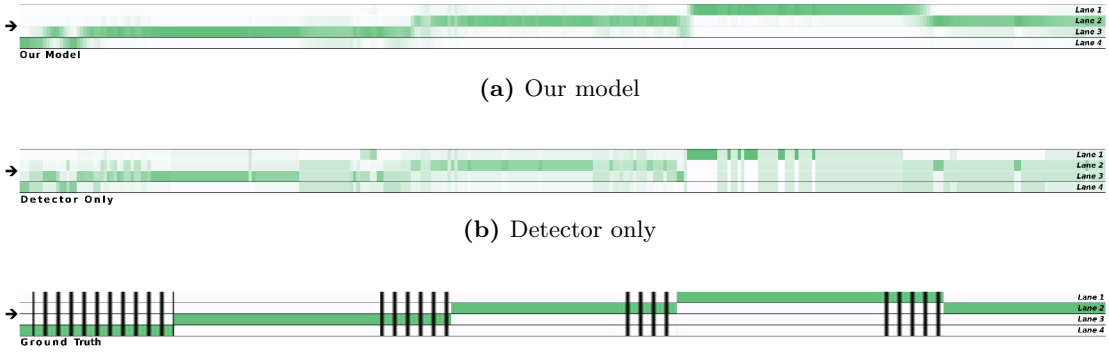
M = mono, S = stereo, L = line, C = clothoid, MLD = [144]

Table 3.6: Parameterization used for the different experimental settings — Spain (3 lanes).

Run ID	Line Detector	σ_1	σ_2	$p1$	$p2$	$p3$	$p4$	BV
06	ML	0.407	0.258	0.692	0.590	0.180	0.459	9
07	SL	0.364	0.460	0.640	0.556	0.409	0.812	8
08	MC	0.313	0.532	0.092	0.255	0.971	0.605	7
09	SC	0.382	0.483	0.941	0.977	0.885	0.984	9
10	MLD	0.343	2.907	0.283	0.991	0.903	0.060	5

M = mono, S = stereo, L = line, C = clothoid, MLD = [144]

Figure 3.9 shows a short area of the A4 highway together with qualitative results of the algorithm, while in Figures 3.10a to 3.10j and 3.11a to 3.11j we report the dispersion over the ego-lane detection, considering all the frames of all experiments. Here the line detectors alone appear clearly unable to correctly detect the ego-lane, mostly because of missing detections due to clutter or illumination issues. This effect often results in the detector not being able to provide any information on the ego-lane (depicted in black in Figures 3.10 and 3.11). Further confirmation of this can be found in Figure 3.9b: the results achieved from relying only on the detector output are extremely noisy, resulting in an unreliable ego-lane determination. For instance, the detector is completely missing the final transition from $Lane_1$ to $Lane_2$, leaving the vehicle without almost any ego-lane localization clue. On the other hand, the filtering effect of the HMM is clearly shown in



(a) Our model

(b) Detector only

(c) Ground truth

Figure 3.9: A short section of the 4-lanes A4 highway in Italy. More saturated colors correspond to a higher probability of being in a lane. The figure presents a comparison between our model (a) with respect to the results achieved using the detector only (b). (c) is the GT, in gray are the transitions between lanes. Our proposal yields good improvements, with more stable detections, compared to the detector's results.

Figure 3.9a. The proposed model correctly identified the lane transitions even without a complete set of line measurements. Promising results are summarized in Figures 3.10 and 3.11. Our model outperformed the basic detector in all tests.

Concerning the confusion matrices Tables 3.7 to 3.26, it is worth noting that, for the dataset recorded in Spain, all the algorithms and configuration settings achieve a better performance compared to the one recorded in Italy. This is most likely related to the better view of the whole road, which contains three lanes instead of four. We can also conclude that the localization results achieved by using our line detector are consistently higher, although this comparison might be unfair for the Italy A4 highway dataset, because of the MLD algorithm limitation (maximum four lines). However, our experiments prove the effectiveness of our proposal, as depicted in Figures 3.10i, 3.10j, 3.11i and 3.11j.

From the result of the experiments, we can observe that:

- using a line detector that is capable of classifying the road line as dashed or continuous results in better performance, see in Figure 3.12 an example of how the performance decreases when not using the Bonus Value.

- In difficult situations that hamper the output of line detection and tracking, our approach is able to provide a valuable performance boost.

Table 3.7: Run#1 - Detector Only

	Actual Class				
Predicted Class	Lane 1	Lane 2	Lane 3	Lane 4	Total
Lane 1	1781	21	5	1	1808
Lane 2	59	1613	190	46	1908
Lane 3	147	42	730	166	1085
Lane 4	1	21	17	215	254
Unassigned	196	754	1310	456	2716
Support	2184	2451	2252	884	7771

Mean Precision: 0.83 Mean Recall: 0.51 Mean F1: 0.61 Accuracy 0.55 Logloss: 0.99

Table 3.8: Run#1 - Model

	Actual Class				
Predicted Class	Lane 1	Lane 2	Lane 3	Lane 4	Total
Lane 1	2079	27	0	0	2106
Lane 2	67	2284	346	67	2764
Lane 3	17	117	1853	409	2396
Lane 4	21	23	53	405	502
Unassigned	0	0	0	3	3
Support	2184	2451	2252	884	7771

Mean Precision: 0.84 Mean Recall: 0.79 Mean F1: 0.80 Accuracy 0.85 Logloss: 0.51

Table 3.9: Run#2 - Detector Only

	Actual Class				
Predicted Class	Lane 1	Lane 2	Lane 3	Lane 4	Total
Lane 1	1941	31	2	4	1978
Lane 2	9	1611	147	31	1798
Lane 3	6	21	762	171	960
Lane 4	3	3	5	223	234
Unassigned	225	785	1336	455	2801
Support	2184	2451	2252	884	7771

Mean Precision: 0.90 Mean Recall: 0.53 Mean F1: 0.64 Accuracy 0.58 Logloss: 0.93

Table 3.10: Run#2 - Model

	Actual Class				
Predicted Class	Lane 1	Lane 2	Lane 3	Lane 4	Total
Lane 1	2117	94	2	0	2213
Lane 2	64	2288	352	49	2753
Lane 3	3	69	1883	385	2340
Lane 4	0	0	15	450	465
Unassigned	0	0	0	0	0
Support	2184	2451	2252	884	7771

Mean Precision: 0.89 Mean Recall: 0.81 Mean F1: 0.83 Accuracy 0.86 Logloss: 0.94

Table 3.11: Run#3 - Detector Only

	Actual Class				
Predicted Class	Lane 1	Lane 2	Lane 3	Lane 4	Total
Lane 1	1438	30	51	0	1519
Lane 2	79	1795	238	124	2236
Lane 3	350	56	1207	269	1882
Lane 4	5	47	33	217	302
Unassigned	312	523	723	274	1832
Support	2184	2451	2252	884	7771

Mean Precision: 0.77 Mean Recall: 0.54 Mean F1: 0.62 Accuracy 0.59 Logloss: 1.07

Table 3.12: Run#3 - Model

	Actual Class				
Predicted Class	Lane 1	Lane 2	Lane 3	Lane 4	Total
Lane 1	1967	69	48	1	2085
Lane 2	85	2268	313	70	2736
Lane 3	104	99	1817	317	2337
Lane 4	28	15	74	496	613
Unassigned	0	0	0	0	0
Support	2184	2451	2252	884	7771

Mean Precision: 0.83 Mean Recall: 0.79 Mean F1: 0.81 Accuracy 0.84 Logloss: 0.63

Table 3.13: Run#4 - Detector Only

	Actual Class				
Predicted Class	Lane 1	Lane 2	Lane 3	Lane 4	Total
Lane 1	1596	27	12	8	1643
Lane 2	20	1910	190	20	2140
Lane 3	11	27	1283	327	1648
Lane 4	7	0	6	206	219
Unassigned	550	487	761	323	2121
Support	2184	2451	2252	884	7771

Mean Precision: 0.89 Mean Recall: 0.57 Mean F1: 0.67 Accuracy 0.64 Logloss: 0.80

Table 3.14: Run#4 - Model

	Actual Class				
Predicted Class	Lane 1	Lane 2	Lane 3	Lane 4	Total
Lane 1	2095	38	0	0	2133
Lane 2	86	2286	240	17	2629
Lane 3	3	127	1907	306	2343
Lane 4	0	0	105	552	657
Unassigned	0	0	0	9	9
Support	2184	2451	2252	884	7771

Mean Precision: 0.87 Mean Recall: 0.84 Mean F1: 0.85 Accuracy 0.88 Logloss: 0.49

Table 3.15: Run#5 - Detector Only

	Actual Class				
Predicted Class	Lane 1	Lane 2	Lane 3	Lane 4	Total
Lane 1	0	0	0	0	0
Lane 2	1	0	45	0	46
Lane 3	39	0	6	0	45
Lane 4	0	0	0	0	0
Unassigned	2122	2413	2171	876	7582
Support	2162	2413	2222	876	7673

Mean Precision: 0.03 Mean Recall: 0.00 Mean F1: 0.00 Accuracy 0.00 Logloss: 1.80

Table 3.16: Run#5 - Model

	Actual Class				
Predicted Class	Lane 1	Lane 2	Lane 3	Lane 4	Total
Lane 1	1120	549	163	59	1891
Lane 2	448	1320	812	160	2740
Lane 3	576	407	1037	515	2535
Lane 4	18	137	210	142	507
Unassigned	0	0	0	0	0
Support	2162	2413	2222	876	7673

Mean Precision: 0.44 Mean Recall: 0.42 Mean F1: 0.42 Accuracy 0.47 Logloss: 1.26

Table 3.17: Run#6 - Detector Only

	Actual Class			
Predicted Class	Lane 1	Lane 2	Lane 3	Total
Lane 1	1405	162	2	1569
Lane 2	68	3004	275	3347
Lane 3	27	6	886	919
Unassigned	479	424	1132	2035
Support	1979	3596	2295	7870

Mean Precision: 0.91 Mean Recall: 0.64 Mean F1: 0.73 Accuracy 0.67 Logloss: 0.73

Table 3.18: Run#6 - Model

	Actual Class			
Predicted Class	Lane 1	Lane 2	Lane 3	Total
Lane 1	1762	263	0	2025
Lane 2	150	3323	326	3799
Lane 3	66	10	1969	2045
Unassigned	1	0	0	1
Support	1979	3596	2295	7870

Mean Precision: 0.90 Mean Recall: 0.89 Mean F1: 0.89 Accuracy 0.89 Logloss: 0.49

Table 3.19: Run#7 - Detector Only

	Actual Class			
Predicted Class	Lane 1	Lane 2	Lane 3	Total
Lane 1	1380	163	2	1545
Lane 2	69	2850	216	3135
Lane 3	29	19	901	949
Unassigned	501	564	1176	2241
Support	1979	3596	2295	7870

Mean Precision: 0.91 Mean Recall: 0.62 Mean F1: 0.72 Accuracy 0.65 Logloss: 0.73

Table 3.20: Run#7 - Model

	Actual Class			
Predicted Class	Lane 1	Lane 2	Lane 3	Total
Lane 1	1749	219	0	1968
Lane 2	155	3322	278	3755
Lane 3	74	55	2017	2146
Unassigned	1	0	0	1
Support	1979	3596	2295	7870

Mean Precision: 0.90 Mean Recall: 0.89 Mean F1: 0.89 Accuracy 0.90 Logloss: 0.48

Table 3.21: Run#8 - Detector Only

	Actual Class			
Predicted Class	Lane 1	Lane 2	Lane 3	Total
Lane 1	1263	82	3	1348
Lane 2	57	2527	208	2792
Lane 3	69	38	748	855
Unassigned	590	949	1336	2875
Support	1979	3596	2295	7870

Mean Precision: 0.90 Mean Recall: 0.55 Mean F1: 0.67 Accuracy 0.57 Logloss: 0.95

Table 3.22: Run#8 - Model

	Actual Class			
Predicted Class	Lane 1	Lane 2	Lane 3	Total
Lane 1	1671	199	49	1919
Lane 2	149	3262	565	3976
Lane 3	158	135	1681	1974
Unassigned	1	0	0	1
Support	1979	3596	2295	7870

Mean Precision: 0.84 Mean Recall: 0.82 Mean F1: 0.83 Accuracy 0.84 Logloss: 0.64

Table 3.23: Run#9 - Detector Only

	Actual Class			
Predicted Class	Lane 1	Lane 2	Lane 3	Total
Lane 1	1375	90	11	1476
Lane 2	46	2882	251	3179
Lane 3	55	19	824	898
Unassigned	503	605	1209	2317
Support	1979	3596	2295	7870

Mean Precision: 0.91 Mean Recall: 0.61 Mean F1: 0.72 Accuracy 0.64 Logloss: 0.79

Table 3.24: Run#9 - Model

	Actual Class			
Predicted Class	Lane 1	Lane 2	Lane 3	Total
Lane 1	1777	166	10	1953
Lane 2	88	3348	363	3799
Lane 3	113	82	1922	2117
Unassigned	1	0	0	1
Support	1979	3596	2295	7870

Mean Precision: 0.89 Mean Recall: 0.88 Mean F1: 0.89 Accuracy 0.89 Logloss: 0.56

Table 3.25: Run#10 - Detector Only

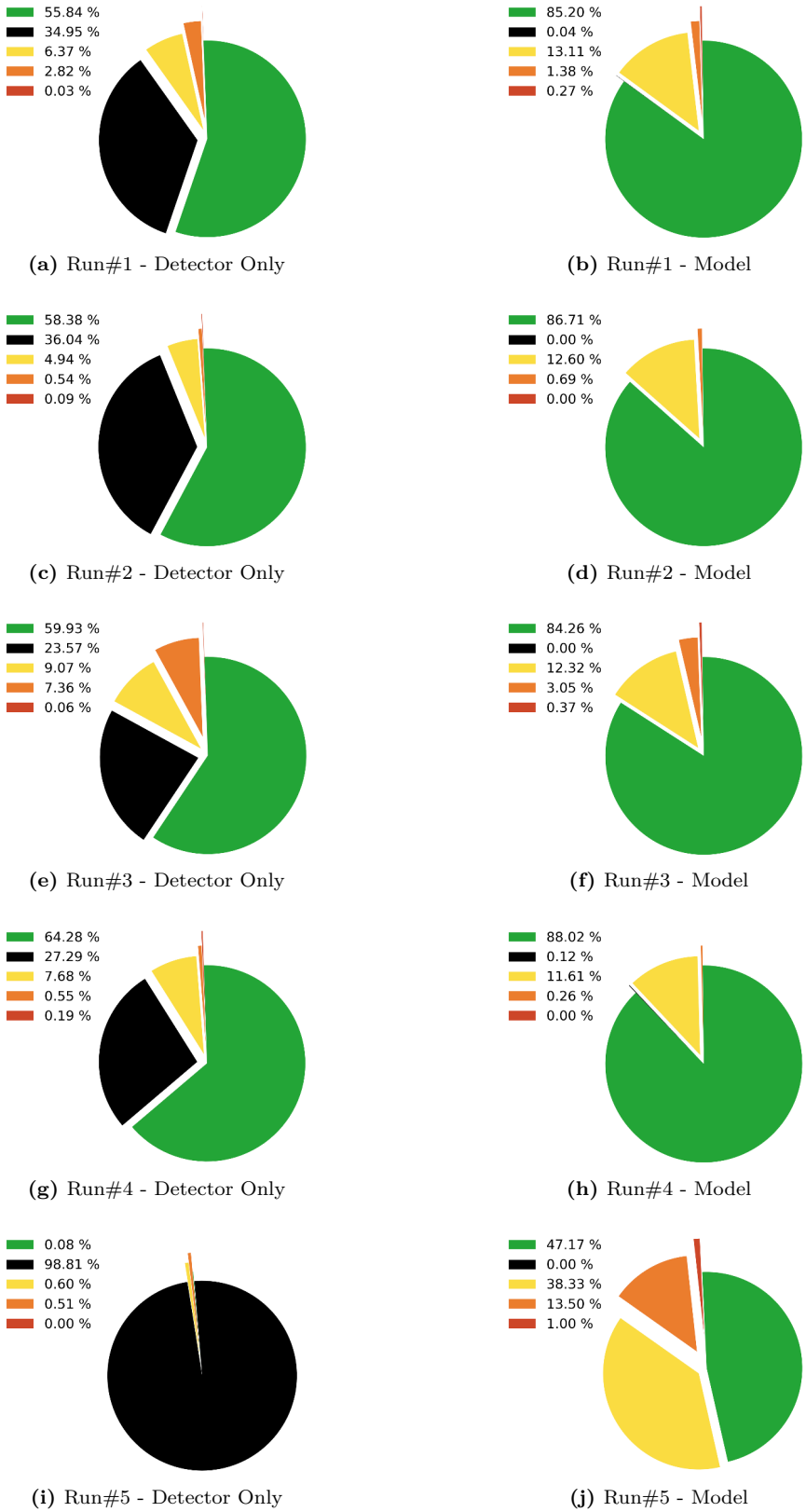
	Actual Class			
Predicted Class	Lane 1	Lane 2	Lane 3	Total
Lane 1	153	104	0	257
Lane 2	58	1714	365	2137
Lane 3	72	138	203	413
Unassigned	1653	1590	1727	4970
Support	1936	3546	2295	7777

Mean Precision: 0.62 Mean Recall: 0.21 Mean F1: 0.29 Accuracy 0.26 Logloss: 1.16

Table 3.26: Run#10 - Model

	Actual Class			
Predicted Class	Lane 1	Lane 2	Lane 3	Total
Lane 1	1436	292	0	1728
Lane 2	317	2841	326	3484
Lane 3	183	411	1969	2563
Unassigned	0	2	0	2
Support	1936	3546	2295	7777

Mean Precision: 0.80 Mean Recall: 0.80 Mean F1: 0.80 Accuracy 0.80 Logloss: 1.08



(a) Run#1 - Detector Only **(b)** Run#1 - Model

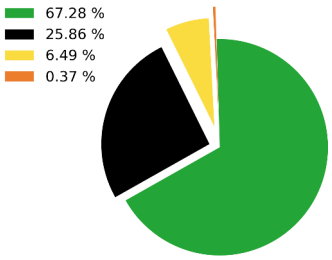
(c) Run#2 - Detector Only **(d)** Run#2 - Model

(e) Run#3 - Detector Only **(f)** Run#3 - Model

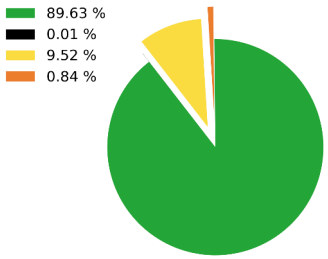
(g) Run#4 - Detector Only **(h)** Run#4 - Model

(i) Run#5 - Detector Only **(j)** Run#5 - Model

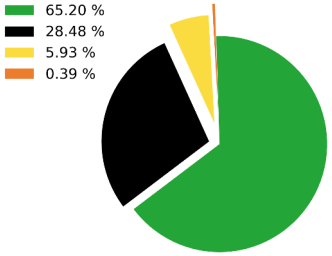
Figure 3.10: Comparison graphs between the localization accuracies using only the line detector compared to the proposed model. As line detector we considered different variants of the "simple" detector described in Section 3.2.1, and the MLD detector. The charts depict the results obtained on the Italian dataset in the A4 highway (4-lanes highway). A green color coding represents correct vehicle's lane localization, yellow represent one-lane range mismatches, orange 2-lane range mismatches and red 3-lane range mismatches. The black color represents the inability to assign a lane, due to missing information.



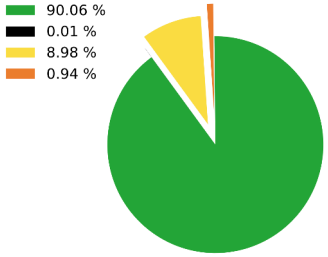
(a) Run#6 - Detector Only



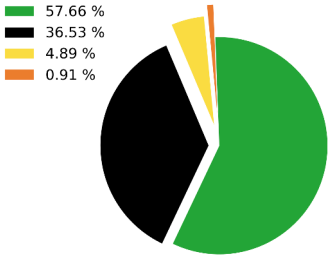
(b) Run#6 - Model



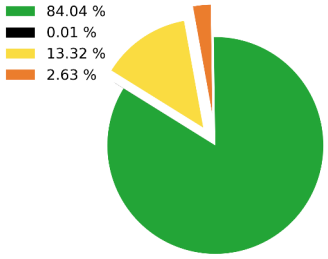
(c) Run#7 - Detector Only



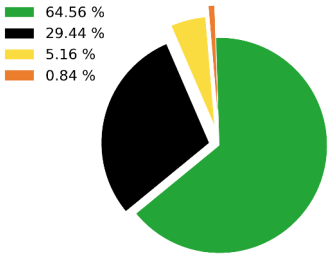
(d) Run#7 - Model



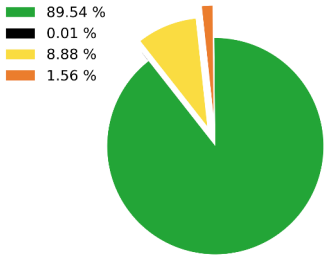
(e) Run#8 - Detector Only



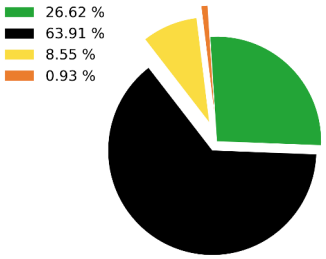
(f) Run#8 - Model



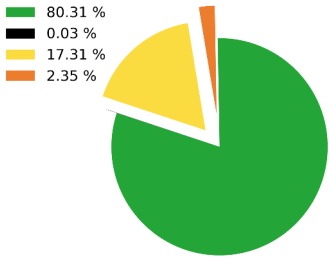
(g) Run#9 - Detector Only



(h) Run#9 - Model

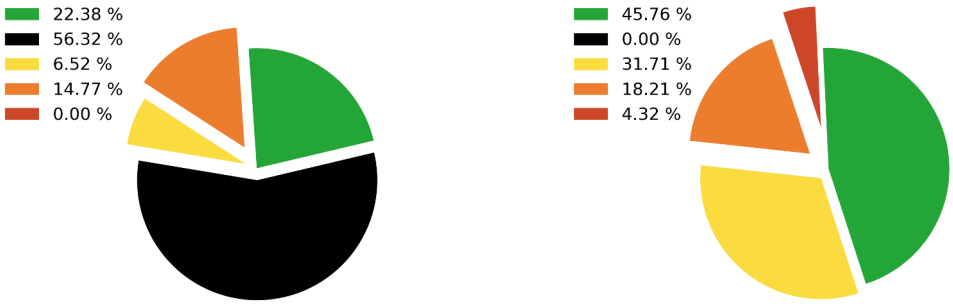


(i) Run#10 - Detector Only



(j) Run#10 - Model

Figure 3.11: As in Fig. 3.10 except it refers to the Spanish dataset, taken in the A2 highway (3-lanes highway).



(a) Run#3 (No BV) - Detector Only

(b) Run#3 (No BV) - Model

Figure 3.12: Comparison of Run#3 (see Figs. 3.10e and 3.10f) between the localization accuracies without the BV, *i.e.,* not classifying the road lines as dashed or continuous.

3.4 Conclusions

In this chapter, we presented an ego-lane estimation algorithm aimed at enhancing the accuracy of the vehicle localization at the lane-level, in highway-like scenarios. Differently from other works in the literature focused on improving the detection of the road lines, we proposed a method designed to cooperate with existing multi-line detectors and trackers. Compared to the state-of-the-art ego-lane estimation literature, our algorithm achieves good localization even when fed with noisy and occasionally missing data, *i.e.,* the typical output of a real, and therefore unreliable, line detector and tracker. We exploited a HMM-based scheme to take advantage of real road line observations probabilistically. The proposed algorithm improves the localization robustness in realistic challenging conditions where lane markings are missing, hidden by traffic clutter or difficult to detect because of lighting issues. Moreover, to validate our approach, we collected and manually labelled a novel dataset composed of more than 100 km of highway driving in both Italy and Spain. In order to allow other researchers to compare their approaches with our results, we published the recorded dataset online.

Chapter 4

Visual localization at intersections

In the previous chapter, we introduced a method to estimate the lane in which the car is driving. Such ego-lane estimation is useful in highway-like scenarios, where the complexity of the environment is limited, *e.g.,* no pedestrians or bicycles, no road intersections, clear lane markings. In urban environments, however, accurate metric localization is vital for an autonomous vehicle, as a wrong position estimate can lead to fatal accidents. Moreover, GNSSs are even more unreliable in urban contexts, where buildings (in full or in part) block or reflect the signals from satellites.

In such contexts, a common approach is to leverage digital maps services. These are usually coupled with sensor fusion methods aimed at exploiting Inertial Navigation Systems (INS) together with the road graph [147, 148], using *lock-on-road* procedures or road geometry modelling approaches [149]. Further improvements are often achieved by computer vision algorithms, which exploit structured urban features, whose position is known in the maps, such as, *e.g.,* road markings [150], or traffic lights [151]. Typical algorithms involve Conditional Random Fields (CRF), Decision Trees or Description Logic techniques [152–154], embedded into sensor fusion frameworks [155, 156]. However, despite the great efforts made to exploit heterogeneous contextual cues [125, 157], a proper localization accuracy is still often difficult to obtain, and this is even truer when approaching and passing through intersection areas.

In urban environments, road intersections represent one of the most critical and dangerous situations for autonomous vehicles due to the complexity of the scene. Car accidents often occur at intersections because they are the most common situation where two or more vehicles try to occupy the same space at the same time. It was estimated that about 40 percent of all car accidents in the US are intersection-related and that the main reason for such crashes is inadequate surveillance from the drivers [158]. Al-

though rear-end collisions are the most common type of car accidents, collisions at road intersections involve more dangerous types of crashes. Broadside collisions (also called T-bone collisions) happen when the front of one vehicle impact the side of another, and they can be fatal for the passengers of the impacted vehicle.

A self-driving car must take into consideration multiple factors when approaching a road intersection — traffic lights, traffic rules, pedestrian crossing — in order to cross it safely. Therefore, accurate and reliable localization is crucial in such situations.

In this chapter, we propose an online vision-based approach for vehicle's localization when approaching road intersections. We exploited a stereo camera configuration mounted on the car to estimate a bi-dimensional probabilistic occupancy grid of the surrounding environment. The localization is then performed against topological information by comparing the occupancy grid generated from the sensors with an "expected" occupancy grid created basing on the road graph geometry retrieved from the OSM service. State-of-the-art CNN-based techniques are employed for both semantic understanding and geometric reconstruction of the scene. While road surface markings can be used to retrieve basic information about the intersection, and then higher-level features such as intersections boundaries or patterns of traffic may be exploited to infer the intersection structure [159], we exploit stereo reconstruction and pixel-level semantic segmentation for this scope. One of the main differences that distinguish our method from other similar localization approaches such as [150, 160], is that our pipeline does not rely on the visual detection of predefined road landmarks (*e.g.,* traffic signs, road markings, stop lines), but rather on a semantic template matching against the cartography. By avoiding relying on such landmarks, our method gains robustness to missed detection and to cluttering elements, situations that often happen in real road driving images.

In order to evaluate the performances of the proposed approach, we performed an extensive analysis of the available datasets for autonomous driving to assemble a new dataset, specially designed for benchmarking localization at intersections. Experimental activity on challenging scenarios demonstrate the effectiveness of our method. Moreover, we define a new evaluation criteria to asses how well the method recovers from a wrong initial position estimate.

4.1 Related Work

The first studies on intersection detection date back to 1987 [161], when Kushner and Puri proposed to determine the intersection geometry by using template matching against an *a priori* map database. Differently from them,the authors in [162] proposed SCARF, a method to exploit road-surface detection and road-type matching, and introduced an intersection model, so to have a system working even when lane markings are missing or under difficult shadow conditions. This system was able to detect intersections without *a priori* knowledge. In [163], the same authors enhanced their work by introducing an intersection detector, based on the ALVINN neural network. Both SCARF and ALVINN were tested in the NAVLab autonomous vehicles presented in Section 2.1.

More recently, and differently from the approaches mentioned before which aimed at the global detection of the intersection, methods based on specific road features were proposed, *e.g.,* [150] where a probabilistic localization method was based on ego-lane identification and a custom set of visual affordances for the identification of lane delimiters, arrows, and stop markings, which were then exploited to localize the vehicle. Similarly was done by the work in [160], which relied on stop lines and their availability in the maps. The solution was assessed under different weather conditions and for several months, achieving decimeter accuracy. The authors in [159] propose an offline method to estimate the intersection geometry using high-level features such as intersections' boundaries and traffic patterns; here, offline means that the method requires the whole sequence of images up to when the vehicle is inside the intersection before starting the computation. These more recent approaches rely on the detection of specific road features to determine the vehicle's pose.

Some research on road intersection detection from LiDAR data had also been performed. In [164], a set of features is extracted from the beam model and combined with a SVM classifier to solve the road shape classification task. The authors in [165] propose to classify the road type using an ANN. Finally, in [166], the authors localize road intersections and detect roads' orientations using the OSM data as prior knowledge.

Differently from the latter methods, our approach is online (output a localization estimate at each frame) and based only on cheap onboard camera sensors (as this is the focus of this thesis). Moreover, our method does not rely on the detection of specific road features; instead, it leverages pixel-level semantic segmentation and 3D reconstruction.

As stated in Section 2.3.3, nowadays CNN-based approaches outperform other solutions in both pixel-level image classification and geometric reconstruction from stereo images. Therefore, in the method presented in this chapter, we exploit state-of-the-art CNN-based techniques for both these tasks.

4.2 The Sensing Pipeline

The localization process is achieved using the detection pipeline shown in Figure 4.1, which was specifically tuned for intersection areas using a CNN-based approach. To represent the intersection area we use an Occupancy Grid (OG), generated from a Bird's-Eye View (BEV), where the cells hold the probability of belonging to the road surface. Our method aims at accurately localize the vehicle with respect to an upcoming intersection, without relying on any specific road feature. It relies on the alignment of the OG generated from the sensing pipeline, called SENSOROG, against a set of OGs obtained from different hypothesized vehicle poses, called EXPECTEDOGs, see Figure 4.2, generated basing on the OSM road graph. Localization is performed using a particle-filter based algorithm, derived from [167].

4.2.1 Semantic Segmentation

The first stage of the sensing pipeline consists in running a CNN for pixel-level semantic classification on the left image. Both Dilation7 [168] and PSPNet [12] have been evaluated. Dilation7 was chosen because it was already trained on KITTI sequences, and PSPNet because it was one of the best performing networks at that time, according to the CITYSCAPES benchmark [169]. For PSPNet, we used the PyTorch implementation proposed in [170] and fine-tuned it in two successive steps. The first fine-tuning phase was performed using the CITYSCAPES dataset, followed by a second phase on the KITTI official labeling sequences [171]. Training details for PSPNet are reported in Section 4.3.4. The Dilation7 network is able to detect 11 categories, namely: *road, sidewalk, building, car, vegetation, sky, fence, pole, sign, pedestrian*, and *cyclist*. PSPNet, instead, detects 19 categories, namely: *road, sidewalk, building, car, truck, bus, motorcycle, bicycle, wall, fence, pole, traffic light, traffic sign, vegetation, terrain, sky, person, rider*, and *train*. As our system is focused on the localization task, further efforts in fine-tuning or in CNN-related research were not pursued. However, as the proposed

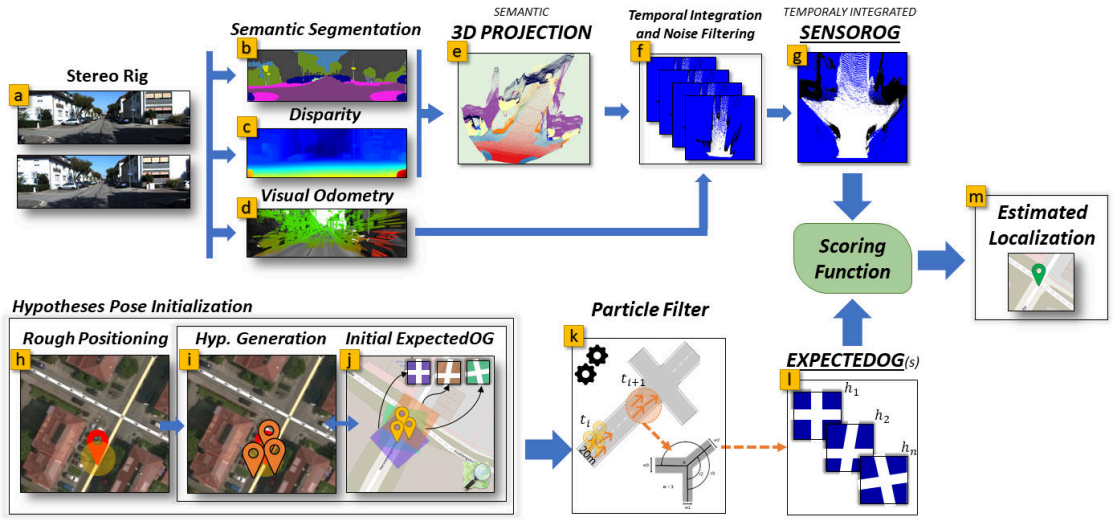
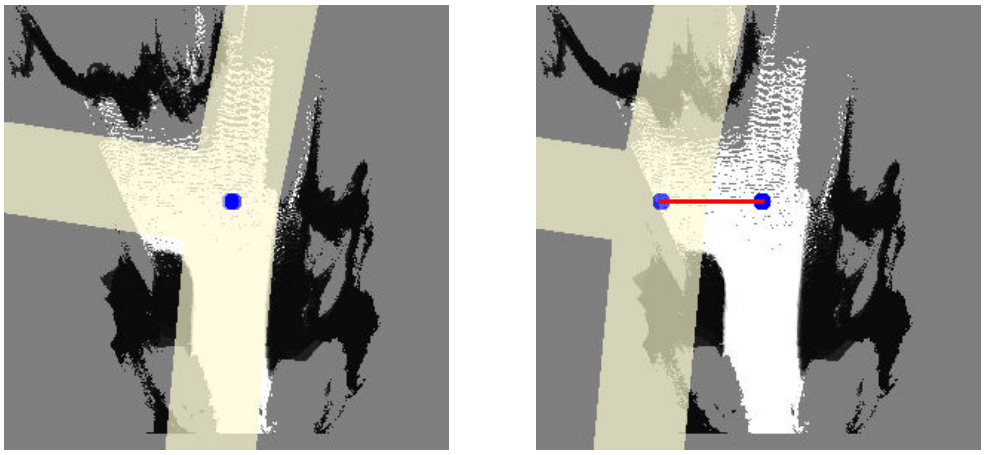


Figure 4.1: From the stereo images (a), the pixel level semantic segmentation (b) and the disparity map (c) are generated, which are then used to create the 3D projection (e). Visual Odometry (d) is used, together with the 3D projection (e), to generate a temporal integrated SENSOROG (g). After the hypotheses initialization (h-j), which is triggered $20m$ from the intersection center, the candidate vehicle poses are filtered using a Particle Filter, thus giving out multiple EXPECTEDOGs (l). A scoring function is then used to evaluate the best match, which corresponds to the best localization estimate (m).

pipeline is not dependent on a specific network, it could easily inherit improvements of future networks. Some semantic segmentation examples obtained with both Dilation7 and PSPNet are reported in Figure 4.3.

4.2.2 3D Reconstruction

A reliable semantic segmentation of an image is, most of the times, not enough to obtain a reliable description of the upcoming road geometry in terms of OGs. Cluttering elements, as well as the distance and the perspective, make identification of the intersection really challenging. For these reasons, notwithstanding the unavoidable stereo errors, we integrated a 3D stereo reconstruction phase. In comparison to the OGs generated from the semantically segmented left image only, this allowed us to obtain OGs that also take into account the geometry of the scene, by just accessing the 3D reconstruction. According to the KITTI-STEREO-2015 benchmark [171], in the last few years, CNN-based approaches have shown top performance in the 3D reconstruction tasks from stereo images. For this reason, we tested the CRL [16] and the PSM [15] networks as they were, at



(a) (b)

Figure 4.2: (a) and (b) presents two hypotheses about the relative pose of the vehicle with respect to the intersection. In beige is the expected intersection geometry, as retrieved from the digital map server, in the two different poses. The sensed intersection is the same for both poses, and it is represented in black and white, plus grey for unknown. In (a) the sensed intersection is almost all covered by the expected intersection. A better match between the sensed and the expected intersection areas corresponds to a shorter distance between the centers of the intersection, *i.e.,* to a better estimate of the vehicle pose.

that time, the two best performing networks in the KITTI-STEREO-2015 benchmark, with the code available. The neural network only computes the disparity, while the 3D reconstruction uses the projection parameters of the cameras.

4.2.3 Generation of the SENSOROG

We have now a classified point cloud, which we project on the ground plane and divide in equally-spaced cells in order to generate the SENSOROG. In particular, the SENSOROG represents an area of 30x30 meters (30 meters forward and 15 meters lateral on each side of the vehicle), and each cell represent a 10x10 cm space, for a total of 300x300 pixels. The values associated with each cell, taken as the probability of being road surface, are computed using the Eq. (4.1), *i.e.,* the ratio of the number of points in the cell classified as *road*, over the total number of points in the same cell.

$$SENSOROG[i,j] = \frac{n_{road}[i,j]}{N[i,j]} \tag{4.1}$$

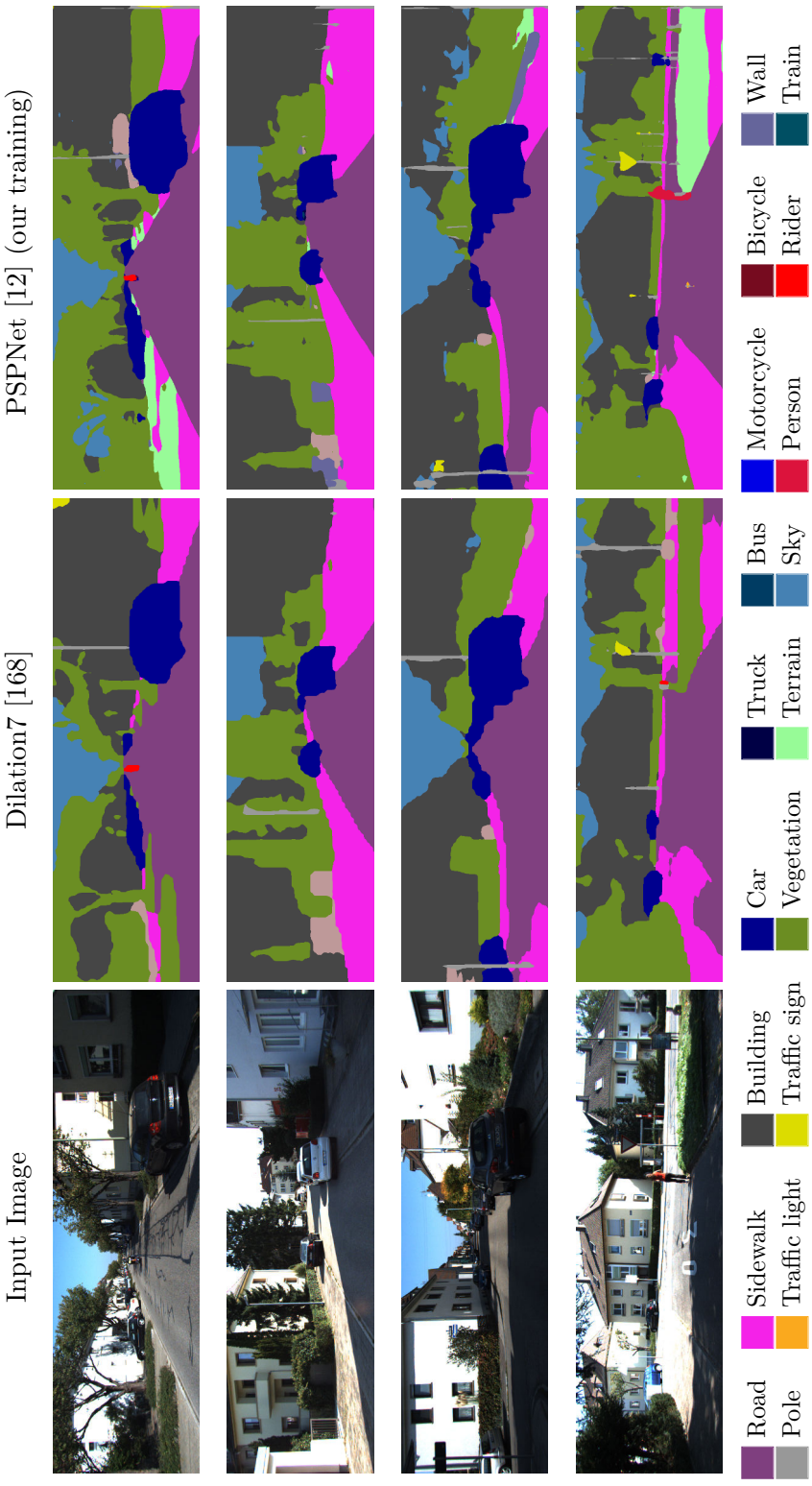
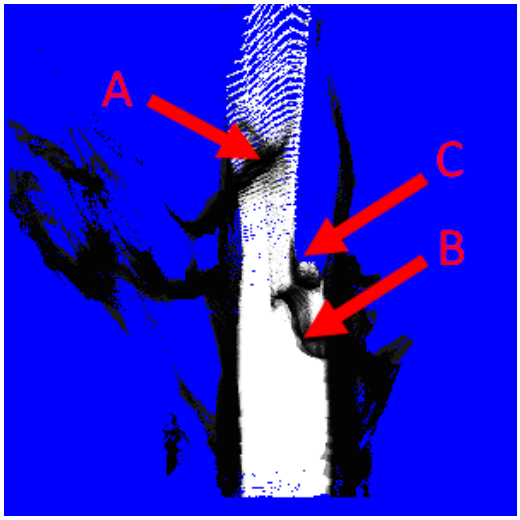
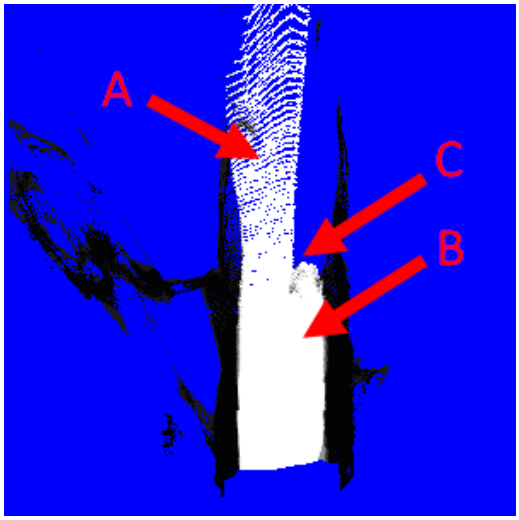


Figure 4.3: Some examples of pixel-level semantic classification on the KITTI dataset, both results from Dilation7 [168] and PSPNet [12] are depicted. Please note that Dilation7 only detects 11 categories, while PSPNet detects 19 categories. For example, PSPNet distinguishes between vegetation and terrain, while Dilation7 does not.

(a)



(b)



(c)

Figure 4.4: The tree branches (A, B) above the road, and the car (C) do not appear in the occupancy grid (c) (heuristics activated) as they appear in (b) (heuristics disabled). In both these figures, a brighter shade of gray corresponds to a higher probability of being road, while the blue is associated to unknown areas.

To increase the quality of the SENSOROG, we first remove points classified as cars, as vehicles may be often found on parking lots or sidewalks, not only on roads. Another issue arise from the presence of trees: besides 3D reconstruction of the tree crowns being very noisy, tree branches often expand above the road surface, and their projections into the SENSOROG cells cause classification mistakes. For this reason, we removed all points positioned higher than a threshold of 2.5 m that were not classified as building. See Figure 4.4 for an impression about the effect of such heuristics.

Even though the SENSOROGs after the latter heuristics are much more reliable, being

able to perceive the whole intersection from a single stereo frame is frequently quite hard, even for human people looking at onboard camera images. Moreover, processing every single frame with the CNN-based classifiers mentioned above usually leads to unstable estimates, thus leading to temporarily unstable occupancy grids. Therefore, we increased the consistency of the SENSOROG by leveraging the outcome computed in consecutive frames, according to the Proportional Conflict Redistribution rule no. 6 (PCR6) derived from the Dezert-Smarandache Theory (DSmT) [172], and updating the vehicle pose with the LIB-Viso2 Visual Odometry (VO) algorithm [173].

In order to further increase the temporal coherence of the SENSOROG, a temporal hysteresis on the classification has been introduced, so that multiple same-class classifications increase the classification belief over each cell of the occupancy grids. We integrate a dual-counter scheme as follows. For each cell, the first counter sums how many times the area has been observed (for many reasons, *e.g.,* occlusions, the 3D reconstruction might not be able to observe points in some area). The other counter starts from the allowed number of consecutive frames that the area will be kept as valid even if not observed, and it is decreased each time the area is not observed. Finally, we reset it when it gets a new observation. If this counter reaches zero, the classification is reset to unknown. The allowed number of consecutive no-observation frames is bounded on both sides (from a minimum of 2 to a maximum of 10); in between, it equals the current value of the first counter. Figure 4.5 depicts the whole scheme.

Finally, in order to remove isolated points that are usually the result of noise in the disparity map, we employed a local spatial filter on the SENSOROG, defined as follows. For each cell, if the total number of points that falls in its 3x3 neighborhood is less than a threshold, that cell is reset to unknown.

4.3 Vehicle Localization

The localization system relies on the quality of the alignment between the SENSOROG and EXPECTEDOGs, which is evaluated using a template matching scoring function. Following a particle filter approach, a set of localization hypotheses is initially drawn, the EXPECTEDOG is then retrieved from the map service, and positioned onto each pose hypothesis. The quality of each alignment correspond to the weight of each hypothesis, and the particle set is then resampled. The new hypotheses are moved forward using the motion estimate derived from a VO module, and then the whole process is iterated.

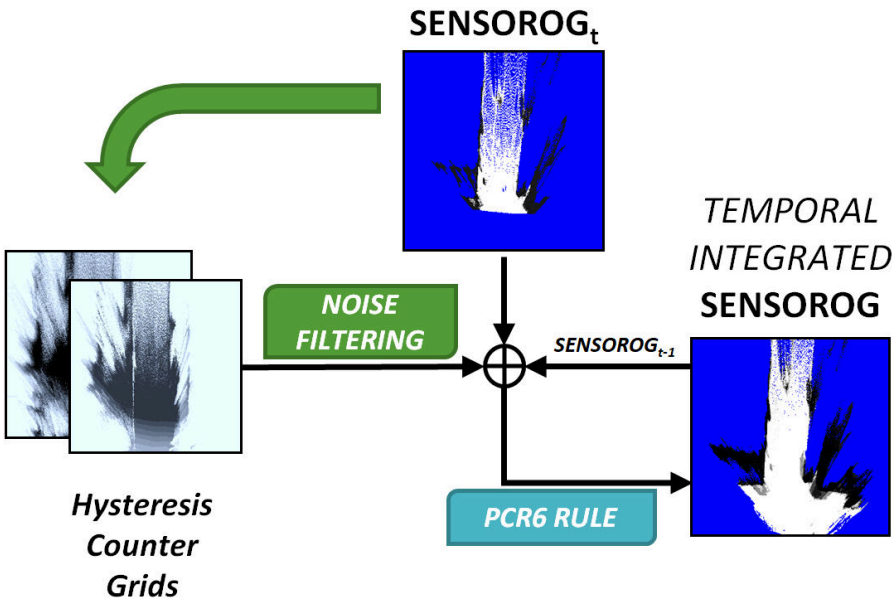


Figure 4.5: The SENSOROG is integrated over time using Visual Odometry and the PCR6 rule, employing a noise filtering procedure through a hysteresis on the classification.

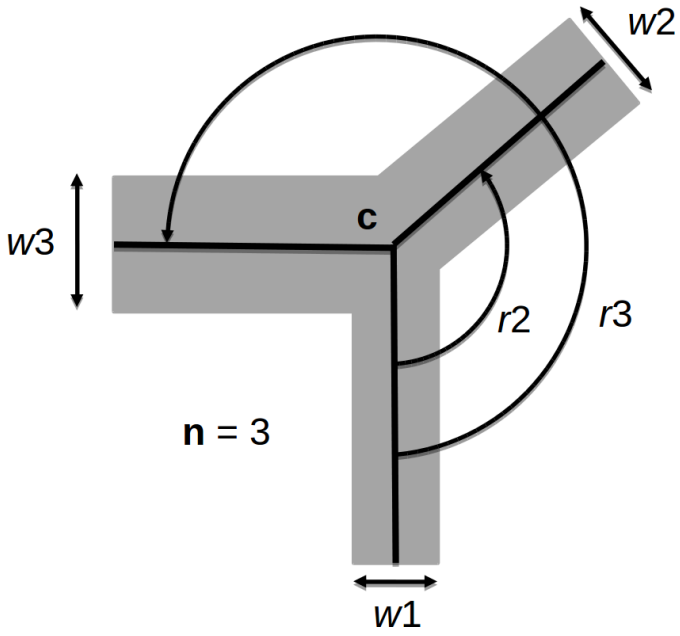


Figure 4.6: The geometry model used to generate the intersections. The parameters c, n, w_i, r_i are retrieved from the OSM service.

4.3.1 Hypotheses Generation

Since we are performing localization with respect to a 2D topological map, the state of each particle space has 3-DoF, *i.e.,* latitude, longitude and yaw, and the parameters of the upcoming intersection model are also included in the particle, relative to each pose hypothesis. Regarding the intersection model, we used the one depicted in Figure 4.6, which has the following parameters:

- The intersection center $\mathbf{c}$ expressed as distance from the vehicle's position
- The number of arms $\mathbf{n}$ involved in the intersection
- The width $\mathbf{w_i}$ and rotation $\mathbf{r_i}$ (expressed relative to the traveled road segment) of each arm i.

As in [163], we initialize the system every time the vehicle gets closer than 20 meters from the intersection center. As we are running on pre-recorded data, we use the GT position (which in KITTI is provided by GPS-RTK) for this trigger, but we mimicked the error that the vehicle might have about its pose in reality, as we will explain in Section 4.4.2. The pose subspace is sampled from a normal distribution whose expected value is the VO output. The EXPECTEDOGs, which represent the hypothesized intersection geometry "as seen from" each particle pose, are then retrieved from the OSM service and added to the state of each particle. Additionally, to accommodate for small cartographic errors as well as unavoidable map simplifications, we slightly modify each EXPECTEDOG by altering the parameters of the model, again sampling from a normal distribution.

4.3.2 Scoring Function

Each particle is then scored by comparing its associated EXPECTEDOG with the SENSOROG, using a template matching function that consists in computing the *Normalized Correlation Coefficient* (NCC). The normalized correlation coefficient between two images I_1 and I_2 is defined in Eq. (4.2).

$$NCC(I_1, I_2) = \frac{\sum_{u,v}((I_1[u,v] - \overline{I_1}) * (I_2[u,v] - \overline{I_2}))}{\sqrt{\sum_{u,v}(I_1[u,v] - \overline{I_1})^2 * \sum_{u,v}(I_2[u,v] - \overline{I_2})^2}} \tag{4.2}$$

Here, $\overline{I_1}$ and $\overline{I_2}$ represent the mean of image I_1 and I_2, respectively. The NCC between each EXPECTEDOG and the SENSOROG can be easily implemented as a 2D GPU-

Convolution using ($SENSOROG - \overline{SENSOROG}$) as the weights of the convolution and ($EXPECTEDOG - \overline{EXPECTEDOG}$) as input. The particle set is updated every frame until the vehicle enters the intersection area. At this point, given the camera configuration and the geometry of most intersections, the OGs is representing an area beyond the intersection, and therefore the filtering is stopped and will be initialized again when the vehicle is near to the next intersection.

4.3.3 Prediction Step

The prediction step of the particle filter is defined as follows.

1. The pose of each particle is updated using a simple motion model that adds the relative displacement obtained from VO and a normally distributed noise, scaled by the magnitude of the displacement. In spite of its simplicity, this proved to work properly, given the relatively short distance traveled by the vehicle in the approach to the intersections.

2. The intersection model of each particle is optionally perturbed, adding a normally distributed noise, in both width and rotation, to each road segment.

4.3.4 Semantic Segmentation — Training Details

We tested four different CNNs, two for the semantic segmentation (Dilation7 and PSP-Net), and two for the stereo disparity estimation (CRL and PSM). Since three networks (Dilation7, CRL, and PSM) were already trained on the KITTI dataset, we did not perform any training activity on these networks. We trained the fourth network (PSPNet) in two consecutive steps, first on the CITYSCAPES dataset, and then fine-tuning on the KITTI dataset.

CITYSCAPES

The images from the CITYSCAPES dataset have size 2048x1024. We employed a pre-processing step to divide each image into 12 overlapping crops of size 800x800, and we treated each crop as an independent image, see Figure 4.7. We trained the network for 100 epochs on a single NVIDIA Titan X, using an SGD optimizer, weight decay of $1 * 10^{-4}$, and a batch size of 2, but updating the network's weights only every six

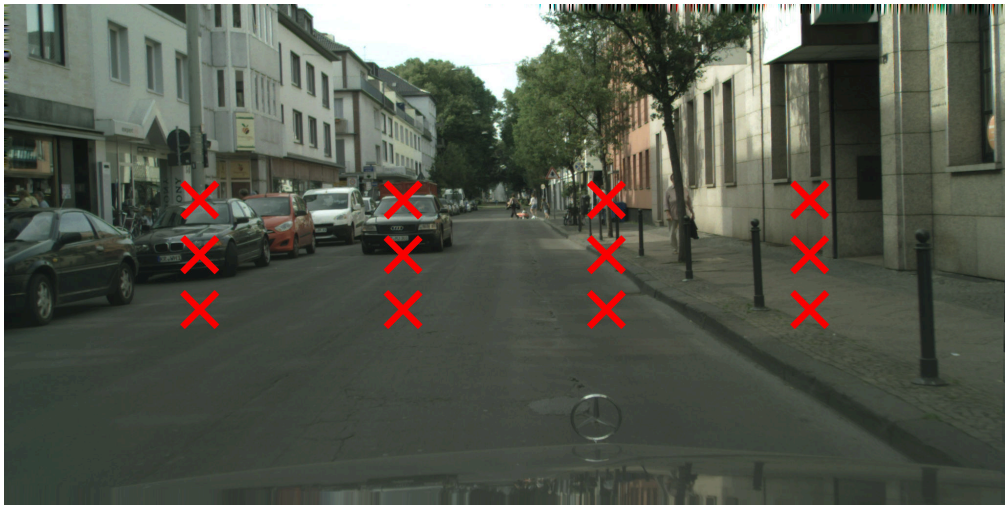


Figure 4.7: A sample image from the CITYSCAPES dataset, with the center of the 12 800x800 crops depicted as red crosses.

forward-backward passes. The initial learning rate was set to 0.01 with a polynomial decay defined as follows:

$$LR(iter) = init\_lr * \left(\frac{1 - iter}{max\_iter}\right)^{0.9} \tag{4.3}$$

Here, *iter* represents the current iteration, while *max_iter* represents the maximum number of iterations.

In order to make the network more robust, we applied a data augmentation scheme by randomly applying rotation $(-10°, 10°)$, horizontal flip, brightness modulation (0.9, 1.1), gamma correction (0.9, 1.1), contrast modulation (0.9, 1.1) and crop (512x512). The loss function used was the one proposed by the PSPNet authors [12], *i.e.,* cross-entropy with an auxiliary loss.

KITTI

After training on CITYSCAPES, we fine-tune the network on the KITTI pixel-level semantic segmentation dataset [171]. Images in the KITTI dataset have shape varying from 1124x370 to 1242x376, and we did not apply any preprocessing cropping in this case. We trained the network for 100 epochs on a single NVIDIA Titan X, using an SGD optimizer, weight decay of $1 * 10^{-4}$, and a batch size of 9. In this case, the learning rate

Table 4.1: KITTI semantic Segmentation — comparison

Method	IoU class	iIoU class	IoU category	iIoU category
APMoE_seg [174]	47.96	17.86	78.11	49.17
SDNet [175]	51.14	17.74	79.62	50.45
PSPNet (fine-tuning)	52.01	24.00	74.91	47.09
SegStereo [176]	59.10	28.00	81.31	60.26
LDN2 [177]	63.51	28.31	85.34	59.07
VideoProp-LabelRelax [178]	72.83	48.68	88.99	75.26

Results on the KITTI test set for different state-of-the-art approaches compared to our training of PSPNet. Even tough the focus of our work was not on the semantic segmentation, we achieved comparable results.

was set to 0.001, with the same decay defined in Eq. (4.3). We applied the same data augmentation scheme used for the CITYSCAPES dataset, except for the cropping, that in this case is of shape 800x256.

The performance of the training on the KITTI dataset have been assessed using the official benchmark website[1]. The results, reported in Table 4.1, are expressed using the Intersection-over-Union (IoU) metric, defined as $IoU = TP/(TP + FP + FN)$, where TP, FP and FN are the numbers of true positive, false positive and false negative, respectively. Some qualitative examples are depicted in Figure 4.3, third column. As the aim of our system is the localization task, additional efforts in semantic segmentation related research were not pursued.

4.4 Experimental Evaluation

4.4.1 Dataset Construction

The obtainment of a set of stereo sequences of a vehicle approaching an intersection, with a reasonably accurate position estimate to be used as GT, has been a very cumbersome activity. Quite unexpectedly, given the number of datasets from road vehicles available today, collecting such sequences was very difficult. We analyzed the sequences of many datasets, checking whether everything needed was usable. Unfortunately, it turned out that, for various reasons, most of the material publicly available was not usable.

[1] `http://www.cvlibs.net/datasets/kitti/eval_semseg.php?benchmark=semantics2015`

After the analysis procedure described below, we ended up with more than 40 approaches to intersections, all from KITTI residential sequences. Although all approaches are from the same German city, we were able to include different intersection geometries, lighting and traffic conditions. In spite of the effort put in building the dataset, and despite its uniqueness, we believe this dataset should be integrated with sequences from other countries, and more extreme light and traffic conditions. Nevertheless, as for today, we believe it is the best mean to benchmark a proposal about vision-based vehicle localization at intersections.

We analyzed the following datasets: KITTI [179], MALAGA [180], Oxford RobotCar [181], Rawseeds [182], and ApolloScapeAuto [183].

- KITTI: used, a stereo rig and GPS-RTK are available;
- Malaga: unused for the lack of GPS-RTK;
- Oxford RobotCar: unused for the lack of GPS-RTK;
- Rawseeds: no intersection between real roads is available, only intersections between roads in private areas, whose resemblance to real intersections has been considered not good enough;
- ApolloScapeAuto: unused for the lack of GPS-RTK.

Unfortunately, we also had to deal with map alignment issues. OSM has global map alignment problems, *i.e.,* some parts of the map are not aligned with each other. The problem manifests with the ground truth trajectory of the vehicle being apparently out of the road, *e.g.,* on curbs or inside private areas. This is a known issue, and there are also literature contributions on how to increase the quality of the alignments, as mentioned in Section 2.2.3. Notice that there is no clear way to determine whether the culprit is the ground truth or the map.

To check this alignment, we started by superimposing the LiDAR point clouds onto the aerial view, by means of the ground truth position, which is available in KITTI for each LiDAR point cloud. Of course, the misalignments between, *e.g.,* the boundaries of the buildings in the aerial view and the same boundaries into the LiDAR point cloud cannot safely be attributed to an error in the ground truth pose or in the aerial map. This superimposition has been performed both with JOSM (an OSM editor), requiring

a cumbersome manual image mosaicing, and also with MAPViz, a ROS tool for superimposing layers of geo-localized data. Unfortunately, in both cases, we could not obtain geometrically consistent results.

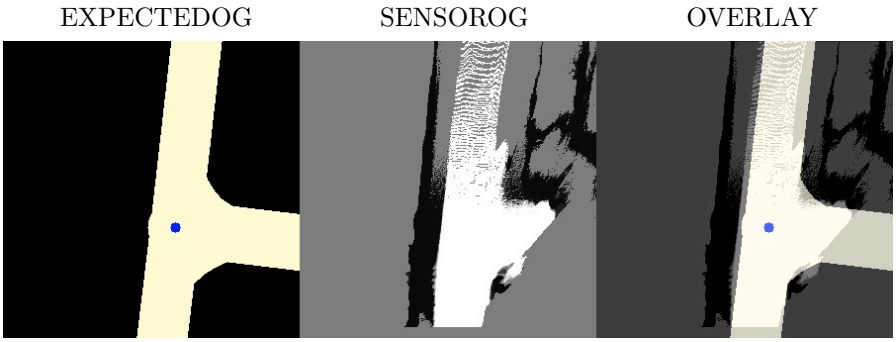
A further option is to use the capability of JOSM to set some alignment points in the aerial images, while the global position of the alignment points could be retrieved from a specific server. At this point, we could modify the ground truths by checking the alignment of buildings and roads. This would have been a very cumbersome and error prone task.

Alternatively, we could join, in a SLAM-like fashion, all aerial images, and then draw the roads and the ground truth position into this map (and re-entry all such data into OSM), again a difficult and very cumbersome task.

In the end, we decided not to try to increase the quality of the maps, neither to correct the ground truth. Instead, by exploiting the spatial locality of the problem (*i.e.,* localization while approaching an intersection), we selected only the sequences taking place in areas where the alignment was good enough, according to the criteria described below. The criteria to select a sequence is as follows.

1. We select a frame where the vehicle is very near to the intersection;
2. we compute the SENSOROG, *i.e.,* the occupancy grid built using the point cloud reconstructed from the cameras (Section 4.2.3);
3. we generate the EXPECTEDOG, *i.e.,* the occupancy grid obtained from OSM, when hypothesizing the vehicle in the ground truth position;
4. we superimpose the two occupancy grids and,
5. depending on whether the two are aligned, see Figure 4.8, we consider the approach to that intersection as usable or not.

Therefore, the localization performances will be benchmarked only in situations where it has been manually verified that the SENSOROGs, are correctly aligned with respect to the corresponding EXPECTEDOGs, computed considering the vehicle in the position ground truth, *i.e.,* a very good estimate of the real position of the vehicle. In the end, we obtained 48 well-aligned subsequences of approaches to different intersection geometries. The involved intersections are shown in Figure 4.9.



(a) Example of correct alignment

(b) Example of wrong alignment

Figure 4.8: For each row, from left to right: the EXPECTEDOG created using the GPS-RTK and the digital map, the SENSORSOG and finally their overlay. A misalignment between the GPS-RTK pose (and the associated EXPECTEDOG) and the map is highlighted in the Figure 4.8b. It is worth to note that alignment error might come from degraded GPS-RTK measures or digital map misalignments, and these two alternatives are not distinguishable.

4.4.2 Evaluation Criteria

We believe that it is necessary to simulate the position uncertainty the vehicle will experience when our algorithm will be triggered. Therefore, we define a new evaluation criteria to asses the capability of the method to recovers from a wrong initial position estimate, that goes as follows:

- The uncertainty on the pose estimate at trigger time is defined a priori and has to be appropriate with respect to the real uncertainties. This uncertainty will also be used online in the vehicle. Hypothesizing this uncertainty to be normally distributed, its 3σ ellipsoid should almost always include the ground truth pose.
- While sampling the hypothesized poses around the initial pose estimate at trigger

Figure 4.9: In the figure the 48 usable subsequences from the KITTI dataset are highlighted in yellow.

time, the ground truth cannot be used as the expected pose, as a real vehicle will have a pose estimate different from the ground truth one.

- To be useful, the experimental activity on pre-recorded datasets should prove that the proposed method can recover from wrong pose initialization.

- The distance of the wrong pose estimate from the ground truth pose should also be realistic, so to have the 3σ ellipsoid, when centered on the wrong pose estimate, to almost always include the ground truth pose, Figure 4.10.

To properly assess whether the method recovers from a wrong position estimate, we could use a randomly distributed or a regularly distributed set of estimates. Given the

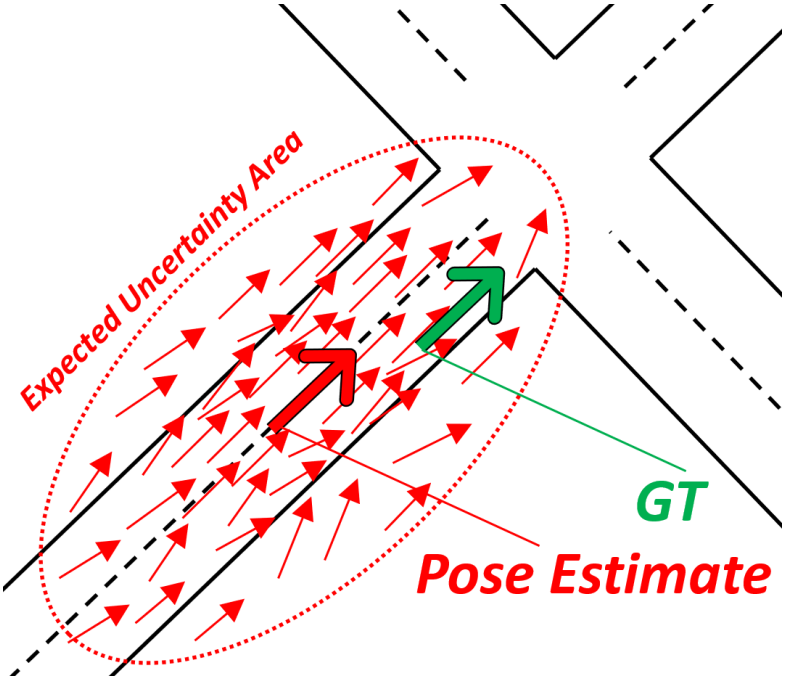


Figure 4.10: Initialization from the pose estimate, which is displaced from the ground truth pose.

seeming smoothness of the convergence behavior, we propose to regularly sample the position of the vehicle, so obtaining a grid, and activating the method taking, as initial position estimate, a different vertex of such grid. At each activation, an initialization from scratch of the method is performed, and the evolution of the localization is analyzed. The localization error, *i.e.,* the accuracy of the pose estimate, is measured against the same-time ground truth, and is recorded for performance evaluation. We propose to use the last set of pose estimates in order to allow some time/space (20 m to 13 m) for the estimate to converge. The measurement is therefore recorded only for those frames whose ground truth pose is between 13 m and 8 m from the intersection center. At shorter distances from the center, given the focal length of the KITTI cameras, the intersection area is usually out of sight, having the vehicle already inside it. The localization error will then be averaged on the recorded frames. Such average will then be in turn averaged across the 48 different approaches to intersections that are available in the dataset. The whole procedure will be repeated for each vertex of the initialization grid, in order to verify the robustness with respect to wrong pose estimates at trigger time. The overall results consist in the matrix of the average localization errors for each initialization value,

and are reported in Figure 4.11.

4.4.3 System Parameterization

The distribution of poses about the initial position is obtained by sampling from a normally distributed noise centered on the vertex of the initialization grid, using null-mean and $\sigma_x = 1.5m$, $\sigma_y = 1.5m$, $\sigma_\vartheta = 0.1rad$ as standard deviations. No GNSS measurement is integrated after the initialization. To find the pose estimated by the filter, we first ranked the particles according to their weights. Then, to simplify the computational complexity of the pose estimation, *i.e.,* to avoid a particle clustering phase, we averaged the 3-DoF state of all the particles having a weight higher than a threshold th, a parameter determined in order to accept just a desired qth percentage of the pose hypotheses. Concerning the scoring function, we used the normalized correlation coefficient, as described in Section 4.3.2.

We have assessed the best system configuration taking into account the following parameters and algorithms on a subset of the dataset, to reduce the computational effort:

- Regarding the particle filter resampling function, we evaluated the Multinomial, the Residual, and the Stratified resampling functions.
- Regarding the neural networks, we considered PSMNet and CRL for the 3D reconstruction, PSPNet and Dilation7 for the semantic segmentation.
- Regarding the pose predicted by the filter, we evaluated different qth percentages.
- Regarding the intersection model, we evaluated whether to add noise or not to each of the road segment parameters (*i.e.,* width and orientation), during the prediction step.

As a first consideration, the results summarized in the first rows of Table 4.2 show that the Stratified resampling function performs better in all our tests. Regarding the DNNs models, 2^{th} block of rows, we assessed which combination of neural networks produced the lowest Mean Absolute Error (MAE), concluding that CRL performed better than PSMNet and Dilation7 performed better than PSPNet. It is worth to notice that, even if PSPNet proved it can achieve high pixel-level classification accuracies on CITYSCAPES, the low performances obtained in our experiments should not be intended as a proof that

Dilation7 outperforms PSPNet, but rather than, with proper training, PSPNet would probably attain a better localization accuracy. Concerning the *qth* value, we assessed that the best performances were obtained considering the 10% high-scored particles. Regarding the road segments parameters (4^{th} column of the table), the results show that adding noise to the orientation parameters turned into better results, while perturbing the width did not cause a comparable MAE decrease. These results suggest that the widths of the roads in the cartography are usually more accurate than the orientation of the road segments. The best system configuration is highlighted in Table 4.2.

We initially evaluated the localization performances regardless of the number of particles, since the GPU implementation of the scoring function allowed a parallel evaluation of a huge number of hypotheses in constant time. Using all the memory of one NVIDIA GXT1080Ti, we were able to handle approximately 8000 hypotheses, which were evaluated in $0.045s$ excluding the initialization phase. Lastly, once the best configuration of the parameters was found, we tried to reduce the number of hypotheses used in the filter. The results, shown in Table 4.3 and depicted in Figure 4.14, prove that even 500 particles are sufficient to localize the vehicle using the proposed approach. Also this activity has been performed using the same subset of the dataset.

The main results of the method are summarized in Figure 4.11. Not surprisingly, almost sub-meter localization accuracies were obtained while our pipeline was initialized nearby the GT position. Even though the results remain reasonably good at a distance from the GT, a decrease of the performances can be observed. However, as long as the estimated pose remains within a significant distance from the GT, the performance continues to be sub-meter. At even larger distances from the GT, we observe a significant degradation of the performances. We believe that this can be attributed to the interplay between intersection geometry and the usage of a template matching scoring, and an example is visible in Figure 4.12. Here, given the geometry of the intersection, H_1 can iteratively move towards the GT pose, *i.e.,* is in the basin of attraction of the GT pose, while H_2 is not since the score does not change with motion along the horizontal axis.

Figure 4.13 depicts two examples of the proposed approach.

Table 4.2: Parameter Estimation

Resampling	CNNs	qth	Width/Orientation	MAE
Multinomial	PSM/Dilation7	10	YES/NO	0.70
Residual	PSM/Dilation7	10	YES/NO	0.72
Stratified	PSM/Dilation7	10	YES/NO	**0.64**
Stratified	PSM/Dilation7	10	YES/NO	0.64
Stratified	PSM/PSP	10	YES/NO	0.85
Stratified	**CRL/Dilation7**	10	YES/NO	**0.63**
Stratified	CRL/PSP	10	YES/NO	0.77
Stratified	CRL/Dilation7	5	YES/NO	0.70
Stratified	CRL/Dilation7	**10**	YES/NO	**0.63**
Stratified	CRL/Dilation7	20	YES/NO	0.64
Stratified	CRL/Dilation7	40	YES/NO	0.71
Stratified	CRL/Dilation7	10	YES/NO	0.63
Stratified	CRL/Dilation7	10	YES/YES	0.61
Stratified	CRL/Dilation7	10	NO/NO	0.66
Stratified	CRL/Dilation7	10	**NO/YES**	**0.60**

To ensure a fair comparison between the experiments, we used an initial *noisy_pose* set to a fixed value, randomly generated using the procedure described in Section 4.4. Please note that in the second column we indicate respectively the neural network used to perform 3D-Reconstruction and Pixel-level Classification.

Table 4.3: Evaluation by particles number

Resampling	CNN	Particles	Width/Orientation	MAE
Stratified	CRL/Dilation7	8000	YES/NO	0.63
Stratified	CRL/Dilation7	2000	YES/NO	0.63
Stratified	CRL/Dilation7	**500**	YES/NO	**0.63**
Stratified	CRL/Dilation7	250	YES/NO	0.72
Stratified	CRL/Dilation7	100	YES/NO	0.84

The table shows the experiments performed to evaluate the adequate number of particles. Here we use the best parameters configuration highlighted in Table 4.2.

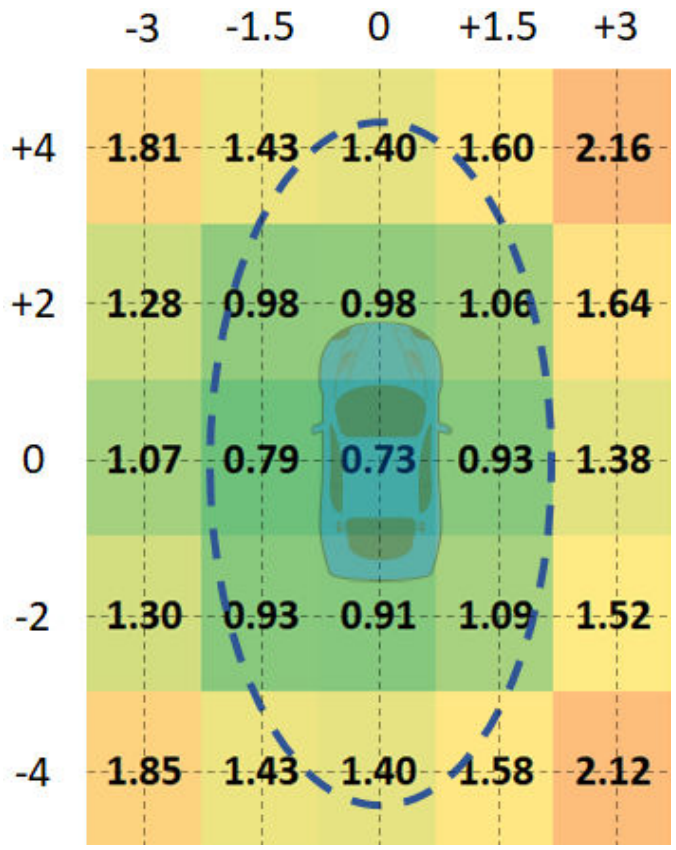


Figure 4.11: The figure presents the average MAE obtained after initializing the proposed method at different initialization values, displaced from the GT.

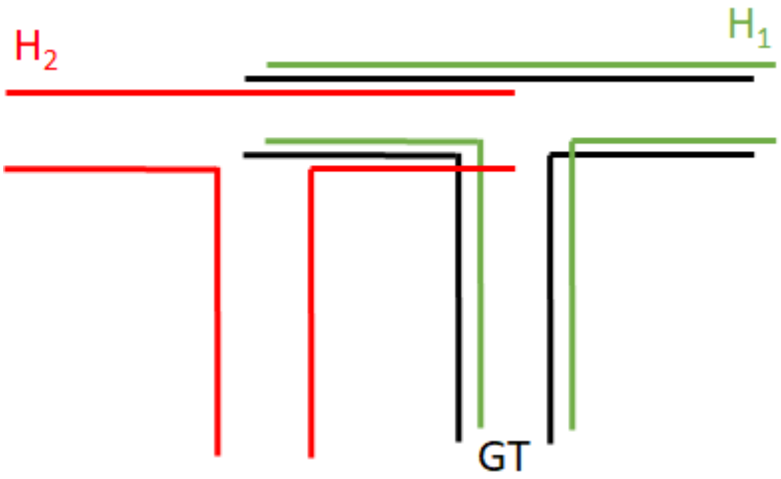


Figure 4.12: In the picture, only the H_1 hypothesis (green) converges to the GT pose (black), because moving H_1 toward the GT increase its score, while moving H_2 along the horizontal axis in either direction does not change its score.

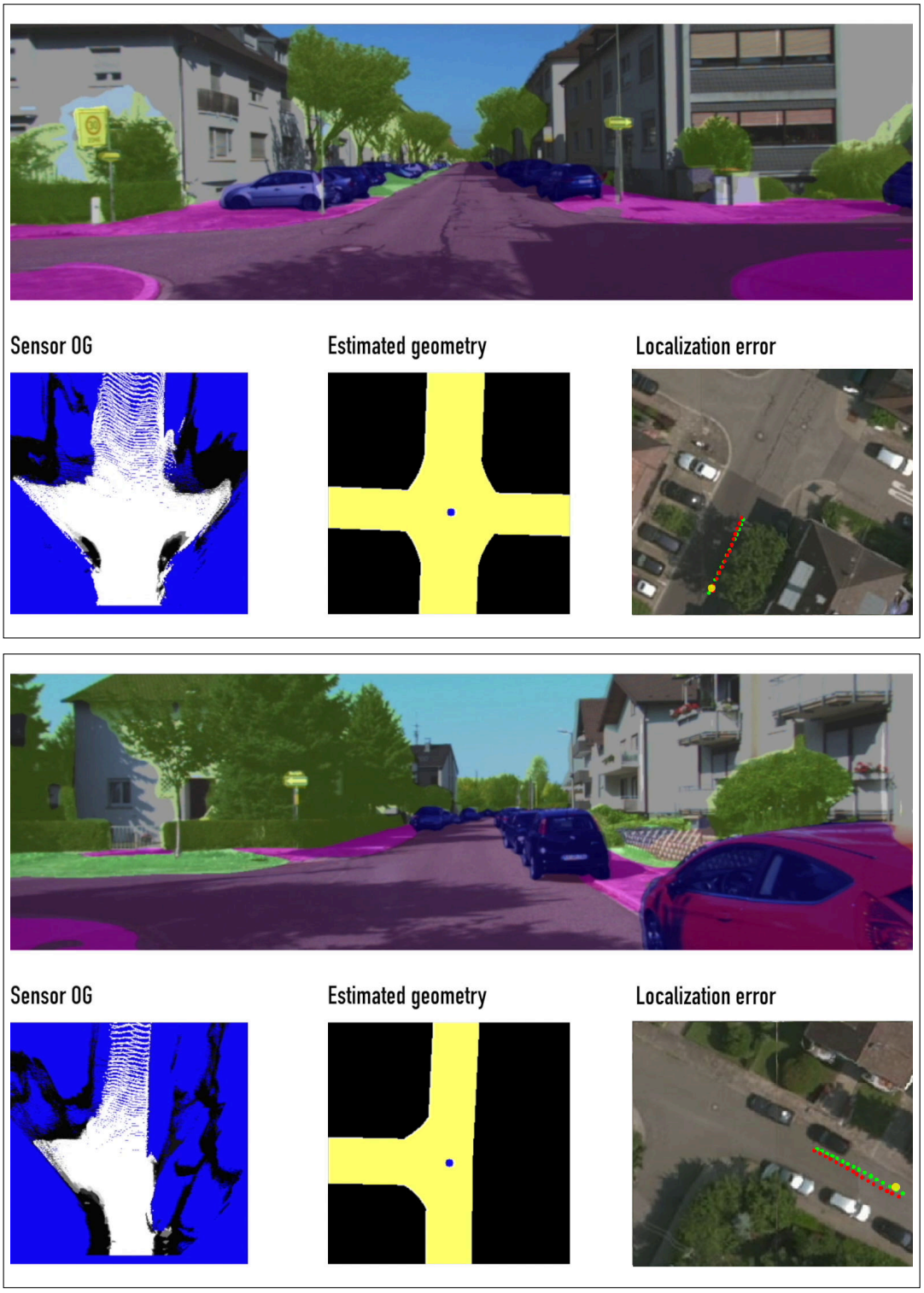


Figure 4.13: Two examples of the proposed approach. Each example depicts the camera image overlaid with the semantic segmentation on top, the SENSOROG on the bottom left, the road geometry associated to the best particle on the bottom center, and the localization error on the bottom right. We compared our estimated localization (red dots) with the ground truth (green dots); the yellow dot represent the initial rough position used to initialize the filter.

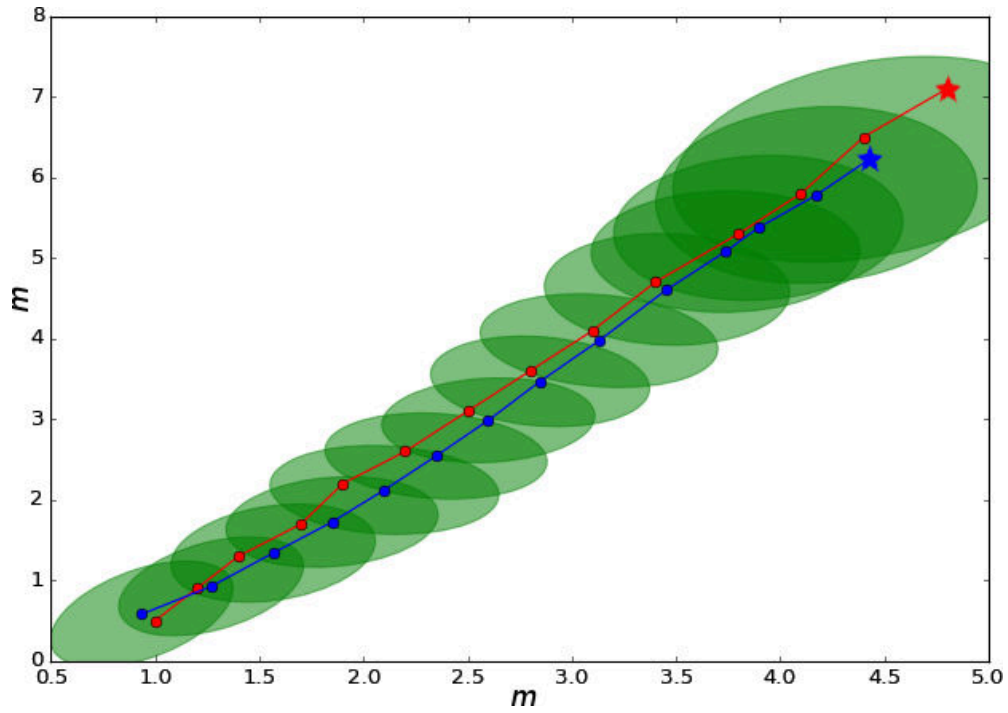


Figure 4.14: An instance of the evolution of the localization estimate (blue) with respect to the GT (red). The colored stars indicate the corresponding starting position. The green ellipses show the 3σ dispersion with respect to the mean value.

4.5 Conclusions

In this chapter, we described a method for estimating the ego-vehicle localization in complex urban scenarios, in particular when the car is approaching intersection areas. Starting from a coarse street-level localization and onboard stereo images, we proposed an original pipeline aimed at exploiting the intersection geometry in conjunction with the output of CNNs. The intersection geometry is retrieved from the OSM service, and the CNNs output represents the perceived surrounding of the vehicle through an occupancy grid. A particle filtering approach was then used to handle the localization uncertainties probabilistically. To asses the performances of the proposed method, we performed an extensive analysis of the publicly available datasets in order to assemble a new dataset for benchmarking vision-based localization methods when the vehicle approach intersections. The resulting dataset consists of 48 sequences, each of which contains an image sequence from a color stereo rig together with an accurate position to be used as ground truth. Results show that our system is able to localize the vehicle up to nearly sub-

meter accuracies starting from poses realistically different from the GT pose. As the map-matching procedure is GPU-performed, our system is able to test a vast number of localization hypotheses. While the obtained results are slightly worse than other approaches to the same task available in the literature, our localization pipeline does not require detecting specific road elements, nor to enhance standard digital maps with precisely localized elements. Specific road data could be, for example, road markings and traffic lights. In our opinion, this is one of the main advantages of the proposed method since individual features might be unreliable in cluttered driving scenarios. For example, differently from state-of-the-art approaches, our method can handle situations where road markings, traffic lights and stop lines are absent or inconsistent such as in road working conditions. Moreover, the system can be easily further improved incorporating semantic considerations in the SENSOROGs, *e.g.,* including moving vehicles and pedestrians detections.

Chapter 5

Enhancing localization by leveraging buildings information

In the previous chapter, we presented an approach for vehicle localization when approaching road intersections. Although intersections are one of the most dangerous scenarios due to the interactions with other road users, in order to localize the vehicle when driving not in proximity of intersection areas different cues should be exploited. In the research community, different approaches have been proposed to localize a vehicle in urban environments by exploiting, for example, road markings [184, 185], stop lines [160], poles [186], or road signs [187]. The latter approaches, however, require a cumbersome prior activity to precisely geo-localize all the road markings, and moreover, they can not give valuable contributions in case of faded or occluded road lines. An interesting opportunity in urban contexts is to exploit the buildings in the vehicle's surroundings, since they are less likely to change over time than, for example, road markings, *i.e.,* buildings are constructed and demolished with low frequency.

It is important to recall that the localization accuracies that can be obtained with GNSSs (such as GPS, Galileo, and GLONASS) are particularly inadequate in urban canyons (*i.e.,* when buildings on both sides flank the vehicle). In such environments, buildings may block the signals from the satellites or deflect them, thereby causing NLOS or multipath errors [2]. Even though different approaches for mitigating the latter GNSS issues by exploiting filtering techniques [30, 31] or by using three-dimensional mappings of the buildings [32, 33] have been proposed, the obtained localization is still not accurate enough for autonomous driving (*i.e.,* they reduced the localization errors from tens of meters to 5-10 meters). A common practice in the autonomous driving community is to exploit a comprehensive understanding of the scene around the vehicle

in order to obtain localization estimates appropriate for safe urban navigation. Even though different techniques have been investigated to improve localization algorithms for self-driving cars [152–154], the results are still inadequate for developing vehicles with driving automation capabilities higher than SAE level 2 (Figure 2.1). Thereby, it remains a remarkable challenge for the research community, as proved by many recent works [59, 63, 188, 189], suggesting the necessity for further improvements.

This chapter presents a probabilistic method for ego-vehicle localization in urban contexts, which exploits the 3D detection of buildings' façades. It bases on the pixel-level stereo image segmentation and 3D reconstruction capabilities of state-of-the-art CNNs. The façades, represented in terms of three-dimensional point clouds, can hence be matched against the surrounding buildings' outlines provided by the OSM service without any preprocessing procedure or map enhancement. Therefore, we can exploit the geometry of the façades, using in our favor the primary source of NLOS or multipath issues, *i.e.,* the buildings themselves. We tested our system on five carefully selected sequences from the KITTI dataset [179], evaluating the localization results with respect to the available GPS-RTK ground truths.

As the outlines of the buildings in OSM are defined by various users and therefore no guaranteed precision is provided, we first manually verified the Karlsruhe urban sequences map alignments, comparing the building data with aerial images and the overlay of the LiDAR data, using their associated ground truth positions. The comparison showed a fairly accurate matching of the building outlines, see Figure 5.1. As a result of this satisfactory alignment, the presented technique achieves meter-level localization accuracy in the context of a vehicle traversing roads with buildings in its field of view, allowing for reliable localization estimates without the typical GNSSs degradation. The use of point clouds generated from the buildings' footprints is a first step towards the use of next generation high-definition maps, which are expected to be very precise and, therefore, will allow for even greater accuracy.

5.1 Related Work

The building detection task from images or point clouds has been widely investigated during the last decades, since it is an important task for applications such as vision-based navigation or mapping. In [190], the authors propose a technique to extract buildings' façades from single images by detecting lines and vanishing points. In [188,

191], a discriminative model was proposed to answer the question: *Is this pixel part of a building façade, and if so, which one?*. They fuse boosting-based pixel-level classification and surface analysis by means of a Markov Random Field. The authors in [192] combined multiple boosted decision trees with the use of auto-context features in order to classify buildings' parts from both images and point clouds.

Recently, DNN-based approaches specially designed for buildings' façades segmentation have been proposed. The method presented in [193] extracts semantic and structural cues with a CNN in order to detect façades. In [194], a semantic segmentation CNN is combined with a Restricted Boltzmann Machine (RBM) to enforce architectural constraints. An interesting approach was proposed in [195], in which two DNNs that jointly process point clouds and multi-view images, using sparse lattice filters, are employed to segment both images and point clouds simultaneously.

All the methods mentioned above for façades detection from images, however, require that the input images depict building in the foreground, and thereby they are not suited for our goal. Images taken from a camera mounted on a vehicle usually depict buildings only on the side of the image. Therefore, for detecting building from onboard cameras, pixel-level semantic segmentation approaches, which nowadays are usually solved using CNNs (Section 2.3.3), are better suited.

Different methods to solve the visual localization task by exploiting buildings have been proposed in the literature. In [196], the authors proposed to detect the silhouette of the buildings present in the scene and register it with a CAD model of the buildings. However, this method requires the buildings to be completely visible in the image. Similarly, the authors in [197] register an image with respect to Geographic Information System (GIS) by matching the skyline, but they provided quantitative results only on synthetic images. The authors in [198] match the buildings detected in an image to the ones present in airborne imagery to perform camera localization; however, they did not provide quantitative results on the localization accuracy achieved.

Matching a reconstructed point cloud to a point cloud obtained from a map service means to register, *i.e.,* to align, the two clouds. This is a well-known problem whose solutions fall into two categories: those that rely on features extracted from the clouds, and those derived from the Iterative Closest Point (ICP) algorithm [199, 200]. The first category is usually aimed at global registration, *i.e.,* at aligning two point clouds without any guess on their initial reciprocal position; this class usually does not provide very

precise results. The second, on the other hand, is aimed at fine-registration, that is, at refining the alignment between two point clouds whose relative pose is roughly known. The main intuition behind the second class of algorithms is that the true point correspondences between the two clouds can be greedily approximated, iteratively looking for closest neighbors. Since in our application the clouds to align are relatively close, *i.e.,* we do not perform localization in global terms, we have chosen a technique from the second category, namely, G-ICP [201], which has been shown to provide very good results [202]. We concentrated only on local registration techniques because our registration problem is inherently local: each vehicle pose hypothesis will have only a small displacement with respect to the real pose of the vehicle, thus the corresponding point clouds should require only a fine alignment with respect to the map. While recent global registration techniques, such as [203], that improve the typical low accuracy of this kind of approach have been proposed, they have not been proved to provide better results than G-ICP with local registration problems.

5.2 Proposed Localization Pipeline

The idea behind the proposed technique is that if a vehicle is well localized, then what it currently perceives should be aligned with the map. This is a well-known concept in the field of robotic localization, therefore we decided to use it to improve the vehicle's localization by leveraging buildings' façades, whose footprints are available in the OSM service. Unfortunately, at the moment, this only means bi-dimensional building outlines, although high-definition maps will likely make a detailed 3D representation of the façades available. Since, while driving, the vehicle usually can observe the buildings in its surroundings, we could align a perceived 3D observation of these buildings with the building models available from OSM. The quality of this alignment is an indirect measure of the quality of the vehicle's localization.

5.2.1 Particle Filter

In order to test the proposed approach, we employed a particle filter derived from [167]. A particle filter is used to estimate the state of a system in a non-parametric way and without making any assumption about the probability distribution representing the state space. In our case, a particle corresponds to a localization hypothesis, and it is associated

Figure 5.1: The figure depicts how we compared the LiDAR data to the aerial images and the OSM data. White points correspond to the LiDAR measures associated to the KITTI ground truth positions.

with a set of scene elements called Layout Components (LCs) that describe the hypothesized surrounding environment. At each iteration of the filter, the particles' position and the associated LCs are updated using the movement predicted by the LIB-Viso2 visual odometry algorithm [173]. Afterward, each particle is given a score, according to the likelihood of the measurement, given the particle's pose. In particular, each LC give its scoring term to the particle, and all the LCs contribute to the final score associated with that particle. Lastly, in the resampling step, particles are randomly taken with a probability proportional to their score.

The basic LC proposed in [167] is based on a *lock-on-road* technique [204], and the score for each particle is based on the distance between the associated pose and its

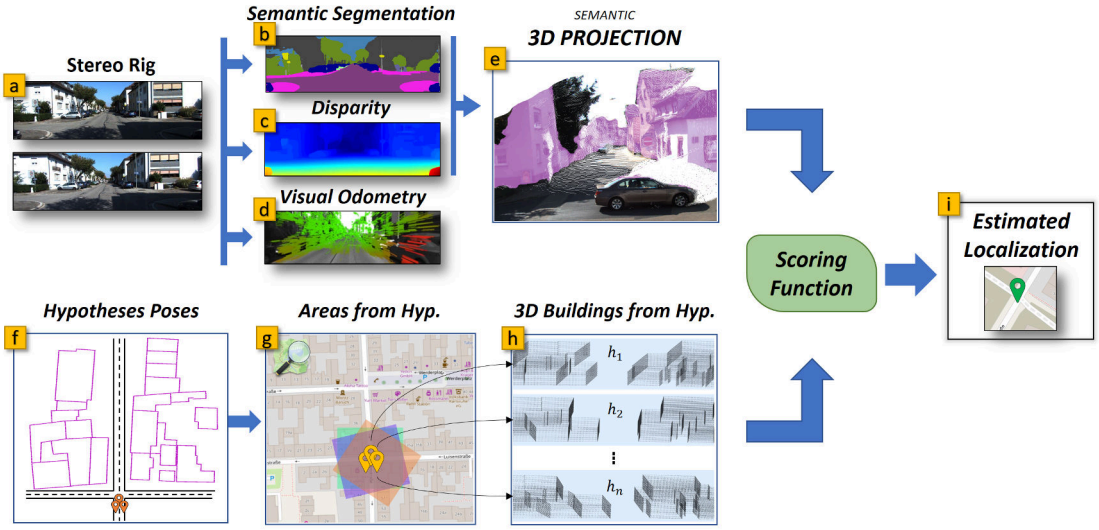


Figure 5.2: The pipeline of our proposal. A stereo rig (a) is used to produce a semantic segmentation (b), the disparity (c) and the visual odometry (d). The disparity and the semantic segmentation are used to produce a point cloud with semantic annotations (e). In the meanwhile, for each vehicle pose hypothesis (particle) we take a surrounding area and the included building outlines (f and g). The outlines are extruded in 3D (h) and compared against the semantic point cloud (e) to produce a scoring for a particle and, at the end, a localization estimate (i).

projection snapped on the nearest road segment in OSM, considering both translation and orientation components. The particle filter with only this basic LC is used as a baseline in our experimental activity. In this work, we propose to enrich the scene description with a new scene element (LC) that takes into account the buildings surrounding the vehicle and uses data from OSM. From a technical perspective, we take the façades of the building as perceived by the vehicle, using onboard cameras, and try to match them with their outlines in OSM converted into point clouds. If the perceived buildings are well aligned with the data in OSM, then the particle represents a good estimate of the pose of the vehicle and therefore should be given a high score. The overall pipeline is summarized in Figure 5.2.

It is important to note that the proposed approach is not meant to be used alone, but it should be combined with other cues to improve the robustness of the approach, *e.g.,* when no building is clearly visible.

5.2.2 The Point Clouds

To represent the façades of the buildings, we decided to use point clouds. The same type of representation has also been used for the outlines extracted from OSM, so that the two data can be aligned using standard point clouds registration techniques.

The point clouds representing the façades have been produced using a CNN fed with images taken from an in-vehicle stereo rig. However, point clouds produced with any sensor could be used as no assumption is made on the characteristics of the point clouds, being it the density, the presence of geometric features, or whether they are organized or not. Nevertheless, we used a stereo rig because this thesis is focused on autonomous vehicles' localization with only cheap camera sensors onboard.

For pixel-level semantic classification, we used again the *Dilation7* network proposed in [168], since it was the semantic segmentation network that achieved the best result in Chapter 4. The semantic segmentation alone is not enough to reconstruct the upcoming road geometry, thus mapping each classified pixel onto the correspondent 3D point is required. Here again, as in Chapter 4, we used the Cascade Residual Learning (CRL) [16] network. However, the core of our approach, that is, the particle scoring method, can also be used in conjunction with other techniques for scene reconstruction and segmentation.

The availability of a semantic segmentation greatly simplifies the alignment of the two point clouds. Indeed, we are able to align the façades of the buildings with the outlines in OSM, removing other elements from the point clouds that otherwise could mislead the alignment step.

The second kind of data that our technique uses are the outlines of the buildings available in OSM. These are simply a list of points describing the corners of a 2D representation of the façades of the buildings. As the observed point clouds are 3-dimensional, while the outlines are bi-dimensional, we have either to project the 3D point clouds onto a 2D plane, thus losing some information as the façades are not just vertical planes, or to extrude the outlines in the 3D space. We opted for the second solution for a very important reason: while still not publicly available, many mapping services (such as Here and TomTom) will probably release soon the so-called HD-maps, which provide a 3D representation of the environment. These representations are also much more accurate than the outlines and also carry much more detail. For example, the balconies cannot be described in a 2D representation, while the vehicle sees them, and they could also provide useful anchoring points for the registration process. Therefore,

our technique is ready to use HD-maps as soon as they become available.

To extrude the outlines, we assumed a fixed height of the buildings. This simplification usually does not bring significant errors because a camera mounted on a vehicle is often not able to observe up to the top of the buildings. While the top of buildings far from the vehicle could still be visible, this data is usually not accurate, since the errors in the stereo reconstruction increase quadratically with the distance. This was also at the basis of using only the point cloud portion closer to the camera. An important advantage of the point clouds produced with the CNN approach mentioned above is that each point is associated with a label describing what that point represents, *e.g.,* a building, the road, a car. In other words, we have a semantic segmentation of every point in the cloud. For our purpose we can therefore discard everything that is not a building, so to avoid aligning other objects with the OSM data and thus improving the results.

5.2.3 The Registration Step

In the field of point clouds registration, the two point clouds are called the source and target point cloud. The pose of the source cloud is iteratively moved to match the target cloud until some convergence criterion is met. We decided to take the distance between the initial pose of the source point cloud (produced by the stereo rig) and its final aligned pose, as a measure of the quality of the vehicle's pose estimate. Intuitively, if the registration process had to move the source point cloud a lot to align it with the target, it means that the pose estimate was worse than if it had to move it by a smaller distance.

Measuring the distance between two poses is a non-trivial problem since there is no shared standard method to measure the combined distance between two positions and two orientations. However, in our case, we can take advantage of the point clouds associated with each iteration step of the registration process. Since we know that the *n-th* point in the source point cloud in the initial pose corresponds to the n-th point in the final pose (rigid-body transformations preserves the order of the points in the cloud), we decided to measure the distance between the two poses by taking the mean distance between all the corresponding points. This is similar to what is done in other fields where a solid is fitted on both poses, and the mean distance between the homologous solid's vertices in the two poses represents the distance between the poses, see [205].

Formally, given two point clouds P and G, that represent the same scene, only dis-

placed, composed of points $p_i \in P$ corresponding to points $g_i \in G$, the distance between the poses of P and G is defined as follows.

$$\delta(P, G) = \frac{\sum_{i=0}^{n} \|p_i - g_i\|_2}{n} \tag{5.1}$$

Here, $\|\mathbf{x}\|_2$ is the Euclidean norm of vector $\mathbf{x}$ and n the cardinality of the point clouds.

To align the two point clouds, we used Generalized-ICP (GICP) [201], a very popular variant of the Iterative Closest Point (ICP) algorithm, which usually provides very good results [202]. The point clouds have been sub-sampled using a voxel grid of a fixed size of 50 cm, to increase the speed of the registration and to reduce the noise introduced during the 3D reconstruction. The point clouds have also been cropped to remove parts too far from the camera (more than 10 m) and, therefore, very noisy. Since we expect the two point clouds to align to be very close, we allowed a maximum of 10 iterations of GICP, which proved to be a satisfactory iteration number in all cases.

Regarding the scoring function associated with the buildings Layout Component introduced in 5.2.1, we need the score to be between 0 and 1; however, the distance between two generic poses obviously does not follow this constraint. Ideally, we wanted the distance to be as close to 0 as possible: this corresponds to the best score for a particle because it means that it is in the right position. Therefore, a distance of 0 should correspond to the maximum score, which is 1. The higher the distance, the lower the score should be, with very large distances corresponding to very low scores. For this reason, to evaluate a particle, we used a Gaussian probability density function (PDF) with zero mean. Then, for a particular value of distance, we take the corresponding normalized value of the density function. This function, indeed, perfectly formalizes our informal requirements for the scoring function.

Summarizing our pipeline, we introduced a new Layout Component in the particle filter presented in Section 5.2.1. The associated scoring factor is the result of the alignment of a point cloud produced with a stereo rig and a point cloud obtained extruding the outlines of the surrounding buildings in OSM. The distance between the initial pose of the particle, and the pose after the alignment, is the new scoring factor: the higher the distance, the farther is the particle from the real pose of the vehicle.

Table 5.1: Experimental results

Sequence	Duration	Length [m]	Baseline Mean Error	Proposal Mean Error
2011_09_26_drive_0005	0:16	66.10	2.043	**0.941**
2011_09_26_drive_0046	0:13	46.38	1.269	**1.168**
2011_09_26_drive_0095	0:27	252.63	1.265	**0.963**
2011_09_30_drive_0027	1:53	693.12	2.324	**1.746**
2011_09_30_drive_0028	7:02	3204.46	**1.107**	1.792

The table show the experimental results on five KITTI sequences. *Baseline* is the particle filter with only the lock-on-road LC, *Proposal* is the filter with also the buildings LC enabled.

5.3 Experimental Evaluation

We tested our approach on the KITTI dataset, composed of several sequences recorded with a vehicle in different environments. Since our approach requires buildings to be present in the surrounding of the vehicle, not all the sequences were appropriate for evaluating the proposed approach. The sequences we used are summarized in Table 5.1. For each sequence, the KITTI dataset provides a ground truth, recorded with an RTK-GPS. Therefore, we measured the quality of our localization estimate by taking, at each time step, the distance with respect to the ground truth position. We compared the results with experiments performed using the particle filter with only its basic *lock-on-road* component, turning on and off the proposed building component. The new proposal got better results in all the sequences but one. While testing the sequence *2011_09_30_drive_0028*, we noticed that there were segments where the ground truth coordinates were wrong: if they were right, the car would have been driving into the buildings. Therefore, we provided results only for the first portion of that sequence, which had a plausible ground truth. With sequence *2011_09_26_drive_0005* the resulting mean error is less than half than the baseline and, very important, has sub-meter accuracy. Sub meter accuracy was also attained for sequence *2011_09_26_0095*, even though the improvement with respect to the baseline is less significant.

On the other hand, the mean error for sequence *2011_09_30_drive_0028* increased. This is probably due to the fact that this sequence has large sections where the buildings surrounding the vehicle are not visible, either because covered by other vehicles, or because there are no buildings at all. Figure 5.3 represents the trajectory for that

sequence: yellow points are the positions as measured by the GPS, while red points are the position estimates of our proposal. As can be seen, our pose estimate diverges from the ground truth mainly in areas where there are few buildings and the vehicle is surrounded by vegetation. Figures 5.4 and 5.5 show two frames taken from a critical section: clearly, no building can be seen. The baseline algorithm without the building component, instead, performed well on this sequence because the car followed more or less strictly the lanes during the recording of the data. This is because the basic *lock-on-road* component aligns the localization hypotheses to the center of the road. An important feature of our new proposal is that, even if the pose estimate is wrong when no building can be seen, the algorithm is able to recover from these *soft* failures as soon as new useful data is received. Indeed, after the first critical section circled in red in Figure 5.3, the pose estimate returns very close to the ground truth.

Since the lock-on-road technique enforces the particles to follow the road graph, in sequences where the car did not follow so strictly the lanes our new proposal performed much better than the baseline. This is the case of sequence *2011_09_26_drive_0005*, whose trajectory is depicted in Figure 5.6. Also in this case there are sections where the error with respect to the ground truth is not negligible. This is probably due to the quality of the point clouds that, even though the buildings are visible, sometimes are very noisy. For example, in Figure 5.7a we have a very noisy point cloud, produced from Figure 5.7b, that corresponds to the area where the proposal got the worst results. As can be seen, the façades of the buildings, although visible, are very noisy and distorted and thus cannot be aligned properly. However, even after such a critical situation, the algorithm is able to recover and to precisely estimate the vehicle's pose, getting much better results than the baseline algorithm, as depicted in Figure 5.8. This is because the vehicle does not strictly follow the road lane, because of the parked truck indicated with a white arrow in Figure 5.6. In these situations, our new *building detection and registration* component adds useful information to the particle filter.

It is important to mention that in its current development state, our proposal does not execute in realtime. To produce the results shown, we played the datasets at half their real frame rate and we used pre-computed point clouds and segmentation. For each frame, on a desktop equipped with an NVIDIA GTX 1080Ti graphics card, calculating the disparity with the neural network took 200 ms, producing a point cloud from the disparity took 20 ms per frame, while producing the semantic segmentation took 500 ms

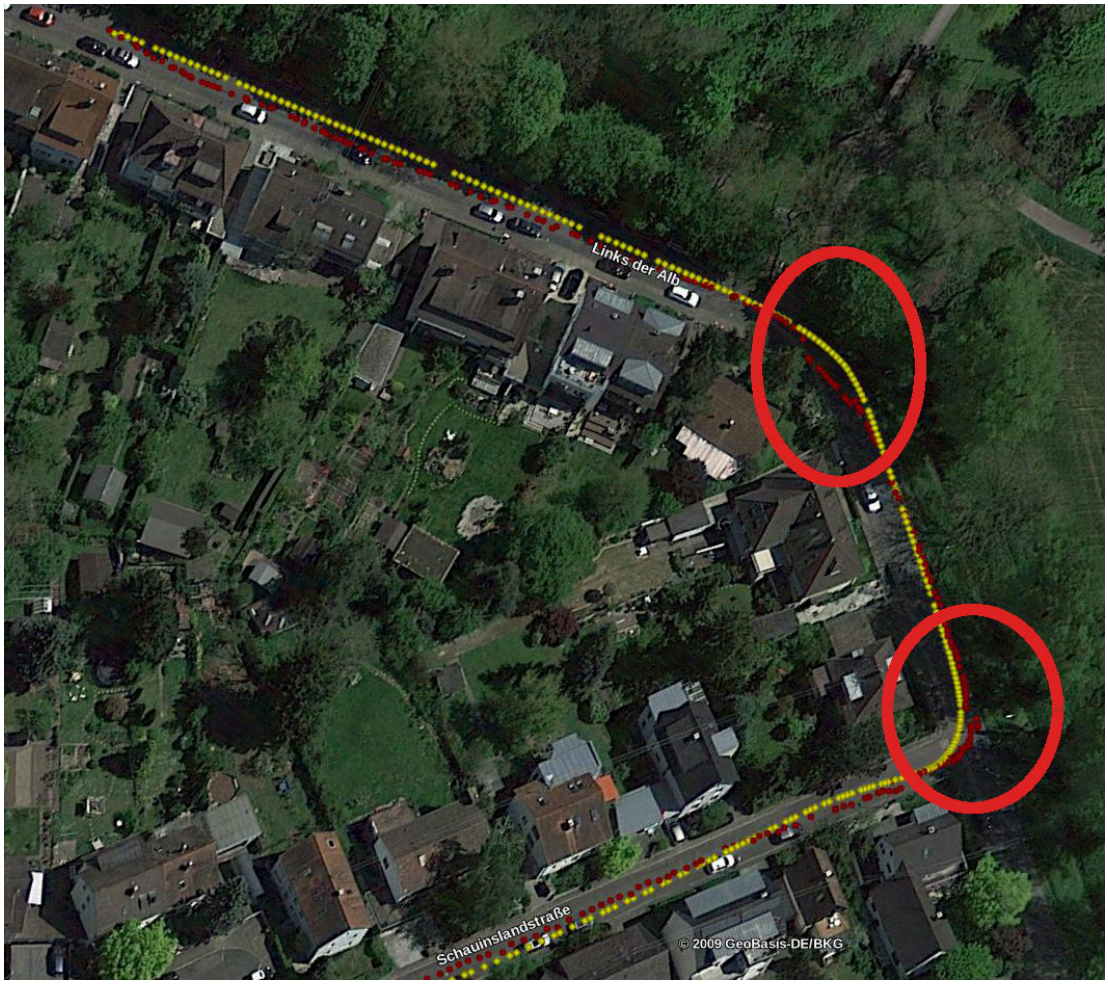


Figure 5.3: The trajectory for the *2011_09_30_drive_0028* sequence. Yellow points are the ground truth positions, red points are the position estimates of our proposal. Critical sections where surrounding buildings are not visible are circled in red.

per frame. It has to be noted that the latter operations, in a real autonomous vehicle, are already needed for other tasks, such as navigation and obstacle avoidance. Therefore, these times are not strictly related to our approach. While the time spent for generating the target point cloud is negligible, regarding the GICP performances our not optimized algorithm took approximately 100 ms using the settings specified in Section 5.2.3.

Figure 5.4: A critical section for our proposal: the vehicle is surrounded by the vegetation.

Figure 5.5: The most critical section for our proposal: the vehicle is surrounded by the vegetation at a turn.

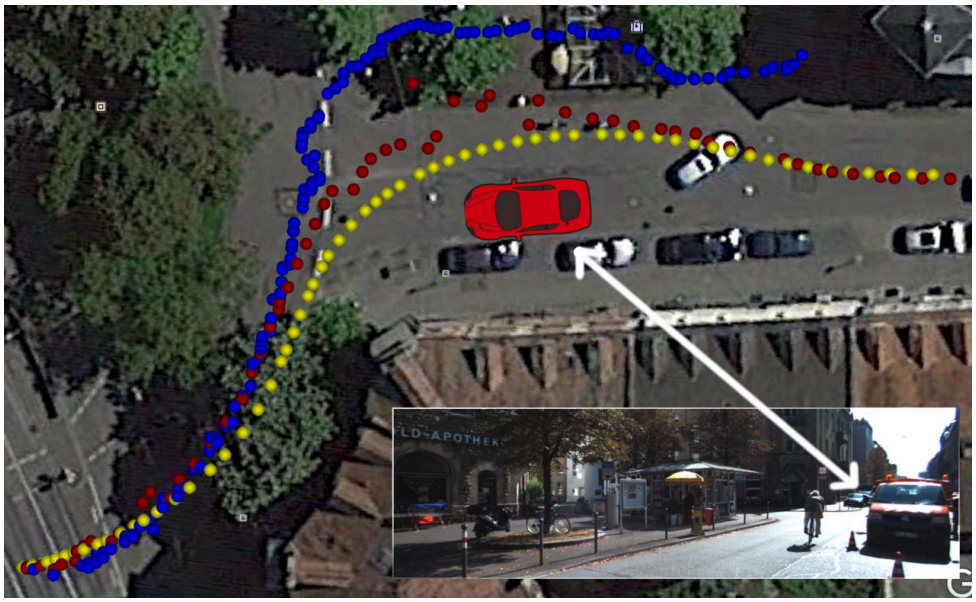
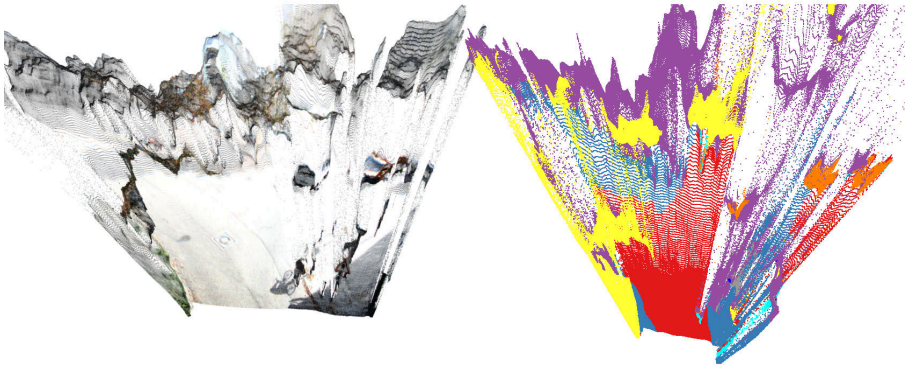


Figure 5.6: The trajectory for the *2011_09_26_drive_0005* sequence. Yellow points are the ground truth positions, red points are the position estimates of our proposal, blue points are the position estimates of the baseline algorithm. The red car is associated to the parked truck visible in the overlaid camera frame.



(a)

(b)

Figure 5.7: A point cloud (a) and the corresponding image (b) from sequence *2011_09_26_drive_0005*. There is too much noise and distortion to obtain a proper alignment with OSM data.

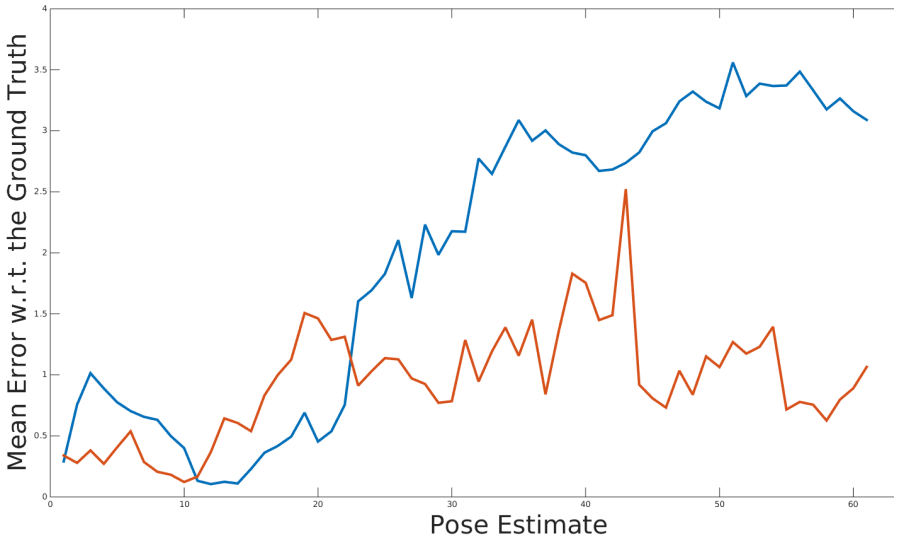


Figure 5.8: The mean error with respect to the ground truth for our new proposal (in red) and the baseline version (in blue) for the *2011_09_26_drive_0005* sequence.

5.4 Conclusions

In this chapter, we presented a novel technique for improving autonomous vehicles' localization in an urban scenario. Our proposal takes into account the buildings in the surrounding of the vehicle and tries to match them with data from OSM. The idea behind this approach is that, if a hypothesized scene correctly represents the current pose, then the façades of the buildings, as seen from that pose, should be aligned with the data in the map. Following this simple idea, we experimented aligning a point cloud obtained from state-of-the-art CNNs with one generated using the outlines of the buildings in OSM. The more we have to move the first cloud, the worse the associated localization hypothesis is. Although we tested the proposed approach with point clouds produced using a state-of-the-art CNN-based technique on stereo images from the KITTI dataset, any sensor that can produce a three-dimensional reconstruction of the environment can be used. Our experiments show that, when buildings are visible from the cameras mounted on the vehicle, the localization accuracy notably benefits from this approach and, as soon as full 3D building point clouds (HD-maps) will be available from mapping providers, our technique will be ready to exploit them.

Chapter 6

Camera to map registration for 6 DoF localization

In the previous chapters, we introduced different approaches for improving autonomous vehicle's localization by exploiting several high-level semantic cues. Specifically, in Chapter 3 we improved the localization in highway-like scenarios by estimating the vehicle's ego-lane, while in Chapter 4 we proposed a probabilistic approach to estimate the upcoming road shape and localize the robot when approaching road intersections. Finally, in Chapter 5, we exploited buildings' façades to mitigate the GNSSs issues in urban canyons (*i.e.,* NLOS and multipath).

A straightforward research direction to complete the works presented in the previous chapters would have been to integrate them in a unified localization framework as well as investigate and model their inter-component interactions. However, Topological map providers usually do not guarantee the localization accuracy of their data. This is even truer for the OSM service, where data are recorded and aligned by the users, often using cheap GPS sensors (*e.g.,* using smartphones), leading to heterogeneous accuracies around the globe. Although different approaches to automatically correct the OSM data have been proposed [55–57], is it still difficult to evaluate the accuracies of the resulting data. Further issues arise from the alignment between topological maps and ground truths provided with autonomous driving datasets; in case they are not properly aligned, there is no clear way to determine which one is incorrect (the map or the ground truths), nor how to align them. As discussed in Section 4.4.1, dealing with the latter issues can be a cumbersome activity.

The considerations mentioned above convinced us to shift our research focus toward the localization against HD-maps, which are specifically designed to support self-driving

vehicles. These maps provide an accurate position of high-level features such as traffic signs and road lanes as well as a representation of the environment in terms of point clouds, with a density of points usually reaching 0.1 m [206]. Companies in the established market of maps and related services, like *e.g.,* HERE or TomTom, are nowadays already developing those maps, which are usually built using LiDAR sensors [206]; this allows other players in the domain of autonomous cars to focus on the localization task. HD-maps, however, are not yet freely available, therefore we used point cloud maps that resemble HD-maps, by processing data from LiDAR sensors using standard SLAM techniques (Section 2.2.1). In the following, we refer to these point cloud maps as LiDAR-maps.

Standard approaches to exploit such LiDAR-maps localize the observer by matching point clouds gathered by the onboard sensor to the LiDAR-map; solutions to this problem are known as point clouds registration algorithms. Currently, these approaches are hampered by the huge cost of LiDAR devices, the de facto standard for accurate geometric reconstruction. However, when these HD-maps will be available, point cloud registration algorithms could be exploited to validate the ground truths of the available datasets and, if necessary, align them with the HD-map of the area.

Differently from standard approaches that localize using onboard LiDAR devices, in this chapter we propose a novel method for registering an image from an onboard monocular RGB camera to a pre-exising LiDAR-map of the area. This allows for the exploitation of the forthcoming market of LiDAR-maps embedded into HD-maps using only a cheap camera-based sensor suite on the vehicle. In particular, we propose CMRNet, a realtime CNN-based approach that achieves camera localization with sub-meter accuracy starting from a rough initial pose estimate. The achieved results show that CMRNet is suitable for accurate localization, even in challenging urban environments. The maps and images used for localization are not necessarily those used during the training phase of the network. To the best of our knowledge, this is the first DNN-based approach to tackle the localization problem without a *localized* CNN, *i.e.,* a CNN trained in the working area [21]. CMRNet does not learn the map, instead, it learns how to match images to the LiDAR-map; thus, it can be used in any environment for which a LiDAR-map is available, without the need to retrain the network. Extensive experimental evaluations performed on the KITTI dataset [115] and on the RobotCar dataset [181] show the feasibility of the proposed approach.

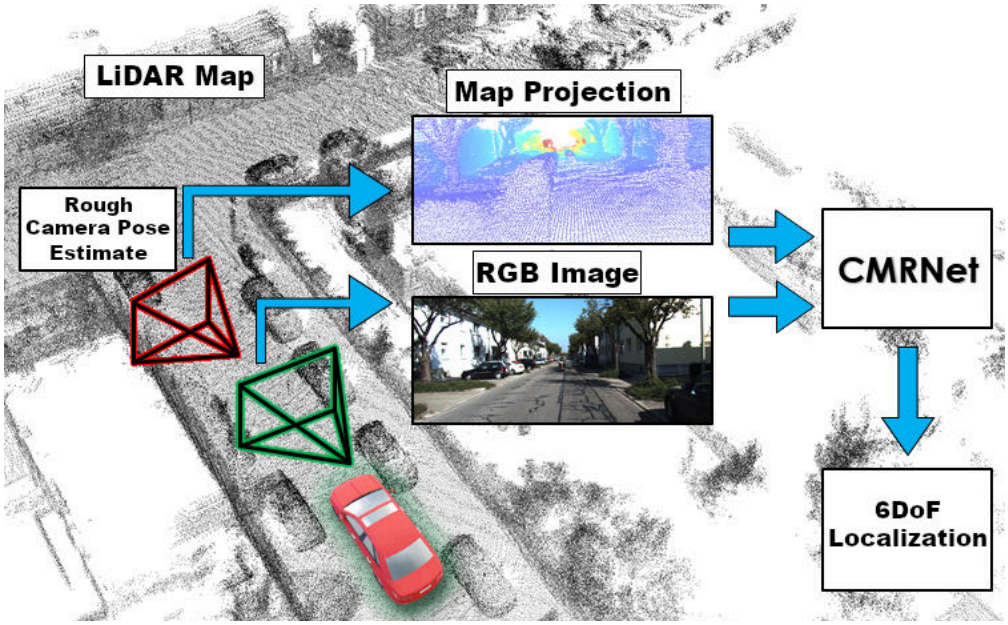


Figure 6.1: A sketch of the proposed processing pipeline. Starting from a rough camera pose estimate (in red), *e.g.,* from a GNSS device, CMRNet compares an RGB image and a synthesized depth image projected from a LiDAR-map into a virtual image plane (red) to regress the 6-DoF camera pose (in green).

The proposed approach takes as input a single RGB image, the LiDAR-map of the environment and a rough estimate of the camera pose (*e.g.,* from a GPS sensor). The output consists of an accurate 6-DoF localization of the camera.

6.1 Related Work

In the last years, visual localization has been a trending topic in both computer vision and robotics communities. Although many localization approaches have been proposed, most of them are either based on images gathered from a camera sensor only, or exploit a map of the environment.

6.1.1 Camera-only Approaches

The first category of techniques deals with the 6-DoF estimate of the camera pose using only images as input. On the one hand, traditional methods face this problem by means of a two-phase procedure that consists of a coarse localization, performed using GNSSs

or a place recognition algorithm, followed by a second refining step that allows for a final accurate localization [207, 208]. On the other hand, the latest machine learning techniques, mainly based on deep learning approaches, face this task in a single step. Models from this second category are usually trained using a set of images taken from different points of view of the working environment, in which the system performs the localization. One of the most important approaches of this category, which inspired many subsequent works, is PoseNet [21]. It consists in a CNN trained for camera pose regression. Starting from this work, additional improvements have been proposed by introducing new geometric loss functions [92], by exploiting the uncertainty estimation of Bayesian CNNs [93], by including a data augmentation scheme based on synthetic depth information [94], or by using the relative pose between two observations in a CNNs pipeline [22]. One of the many works that follow the idea presented in PoseNet is VLocNet++ [23]. Here the authors deal with the visual localization problem using a multi-learning task (MLT) approach. Specifically, they proved that training a CNN for different tasks at the same time yields better localization performances than single task learning. At the time of writing, the literature still sees [23] as the best performing approach on the 7Scenes dataset [209]. In [210] the authors developed a CNN that exploits a sequence of images in order to improve the quality of the localization in urban environments. The authors in [211, 212], instead, integrated a differentiable version of RANSAC within a CNN-based approach in an end-to-end fashion.

Other camera-only localization methods are based on *decision forests*, which consists of a set of decision trees used for classification or regression problems. For instance, the approach proposed in [209] exploits RGB-D images and regression forests to perform indoor camera localization. The techniques mentioned above, thanks to the generalization capabilities of machine learning approaches, are more robust against challenging scene conditions like lighting variations, occlusions, and repetitive patterns, in comparison with methods based on hand-crafted descriptors, such as SIFT [43], or SURF [44]. However, all these methods cannot perform localization in environments that have not been exploited in the training phase, hence these regression models need to be retrained for every new place. Moreover, they achieved good results only in small environments (*e.g.,* small rooms or outdoor areas of few tens of meters). Therefore, despite the great localization performance achieved in small environments, these approaches are not suitable for autonomous driving applications.

6.1.2 Camera and LiDAR-map Approaches

The second category of localization techniques leverages existing maps, in order to solve the localization problem. Since vision-based approaches that try to localize a vehicle exploiting topological maps have been discussed in Chapters 3 to 5, here we focus only on existing methods for visual localization within LiDAR-maps. In particular, two classes of approaches have been presented in the literature for this task: geometry-based and projection-based methods. A geometry-based method that try to solve the visual localization problem by comparing the point cloud reconstructed from a sequence of images and the existing map has been proposed in [213]. The authors in [214], instead, developed a projection-based method that uses meshes built from intensity data associated to the 3D points of the maps, projected into an image plane, to perform a comparison with the camera image using the *Normalized Mutual Information* (NMI) measure. Finally, in [215], the authors proposed to use the similarity between depth images generated by synthetic views and the camera image as a score function for a particle filter, in order to localize the camera in indoor scenes.

The main advantage of these techniques is that they can be used in any environment for which a 3D map is available. In this way, they avoid one of the major drawbacks of machine learning approaches for localization, *i.e.,* the necessity to train a new model for every specific environment. Despite these remarkable properties, their localization capabilities are still not robust enough in the presence of occlusions, lighting variations, and repetitive scene structures.

The work presented in this chapter has been inspired by [216], which used 3D scans from a LiDAR and RGB images as the input of a novel CNN, namely RegNet. Their goal was to provide a CNN-based method for calibrating the extrinsic parameters of a camera with respect to a LiDAR sensor. Taking inspiration from that work, in this chapter we propose a novel approach that has the advantages of both the categories described above, *i.e.,* robustness to visual changes and capability to perform localization in any environment for which a LiDAR-map is available without retraining. Differently from [216], which exploits the data gathered from a synchronized single activation of a 3D LiDAR device and a camera image, the inputs of our approach are a complete 3D LiDAR map of the environment, together with a single image and a rough initial guess of the camera pose. Eventually, the output consists of an accurate 6-DoF camera pose localization. It is worth to notice that having a single LiDAR scan taken at the same

time as the image, as in [216], imply that the observed scene is exactly the same. In our case, instead, the 3D map usually depicts a different configuration, *i.e.,* road users are not present, making the matching considerably more challenging.

Our approach combines the generalization capabilities of CNNs, with the ability to be used in any environment for which a LiDAR-map is available, without the need to re-train the network.

6.2 Proposed Approach

This work aims at localizing a camera from a single image in a 3D LiDAR-map of an urban environment. Toward this goal, we exploit recent developments in deep neural networks for both pose regression [21] and feature matching [17].

The pipeline of our approach is depicted in Figure 6.1 and can be summarized as follows. First, we generate a synthesized depth image by projecting the map points into a virtual image plane, positioned at the initial guess of the camera pose. This is done using the intrinsic parameters of the camera. From now on, we will refer to this synthesized depth image as LiDAR-image. The LiDAR-image, together with the RGB image from the camera, are fed into the proposed CMRNet, which regresses the rigid body transformation H_{out} between the two different points of view. From a technical perspective, applying H_{out} to the initial pose H_{init} allows us to obtain the 6-DoF camera localization.

In order to represent a rigid body transformation, we use a $(4, 4)$ homogeneous matrix:

$$\mathbf{H} = \begin{pmatrix} \mathbf{R}_{(3,3)} & \mathbf{T}_{(3,1)} \\ \mathbf{0}_{(1,3)} & \mathbf{1} \end{pmatrix} \in SE(3) \tag{6.1}$$

Here, $\mathbf{R}$ is a $(3, 3)$ rotation matrix and $\mathbf{T}$ is a $(3, 1)$ translation vector, in Cartesian coordinates. The rotation matrix is composed of nine elements, but, as it represents a rotation in the space, it only has three degrees of freedom. For this reason, the output of the network in terms of rotations is expressed using quaternions lying on the 3-sphere (S^3) manifold. On the one hand, even though normalized quaternions have one redundant parameter, they have better properties than Euler angles, *i.e.,* gimbal lock avoidance and unique rotational representation (except that conjugate quaternions represent the same rotation). Moreover, they are composed of fewer elements than a

rotation matrix, thus being better suited for machine learning regression approaches.

The outputs of the network are then a translation vector $\mathbf{T} \in \mathbb{R}^3$ and a rotation quaternion $\mathbf{q} \in S^3$. For simplicity, we will refer to the output of the network as H_{out}, implying that we convert $\mathbf{T}$ and $\mathbf{q}$ to the corresponding homogeneous transformation matrix, as necessary.

6.2.1 LiDAR-Image Generation

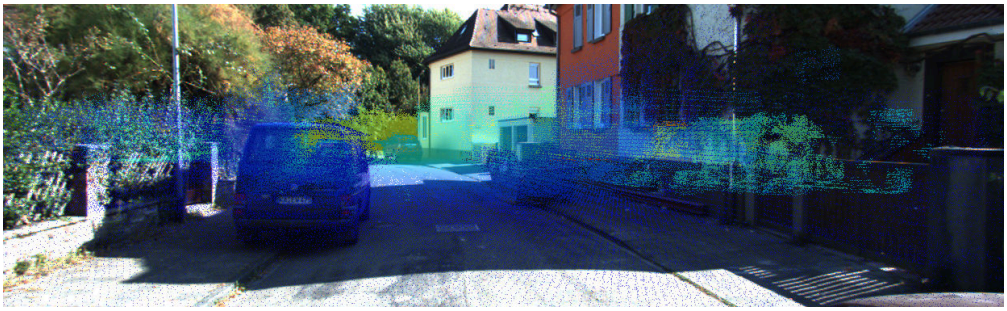
In order to generate the LiDAR-image for a given initial pose H_{init}, we follow a two-step procedure.

Map Projection. First, we project all the 3D points in the map into a virtual image plane placed at H_{init}, *i.e.,* compute the image coordinates p^i of every 3D point P^i. This mapping is shown in Equation (6.2), where K is the camera projection matrix.

$$p^i = K \cdot H_{init} \cdot P^i \tag{6.2}$$

The LiDAR-image is then computed using a z-buffer approach to determine the visibility of points along the same projection line. Since Equation (6.2) can be computationally expensive for large maps, we perform the projection only for a sub-region cropped around H_{init}, ignoring also points that lay behind the virtual image plane. In Figure 6.2a is depicted an example of LiDAR-image.

Occlusion Filtering. The projection of a point cloud into an image plane can produce unrealistic depth images. For instance, the projection of occluded points, *e.g.,* laying behind a wall, is still possible due to the sparsity nature of point clouds. To avoid this problem, we adopt the point clouds occlusion estimation filter presented in [217]; an example of the effect of this approach is depicted in Figure 6.2b. For every point P_i, we can build a cone, about the projection line towards the camera, that does not intersect any other point. If the cone has an aperture larger than a certain threshold Th, the point P_i is marked as visible. From a technical perspective, for each pixel p_j with a non-zero depth in the LiDAR-image, we compute the normalized vector $\vec{v}$ from the corresponding 3D point P_j to the pin-hole. Then, for any 3D point P_i whose projection p_i lays in a neighborhood (of size KxK) of p_j, we compute the vector $\vec{c} = \frac{P_i - P_j}{\|P_i - P_j\|}$ and the angle ϑ between the two vectors $\vec{v}$ and $\vec{c}$ ($\vartheta = \arccos(\vec{v} \cdot \vec{c})$). This angle is used to assess the visibility of P_j. Occluded pixels are then set to zero in the LiDAR-image. More details



(a) Without Occlusion Filter

(b) With Occlusion Filter

Figure 6.2: Top: a LiDAR-image with the associated RGB overlay. Please note how the points behind the building on the right, *i.e.,* lighter points on the fence, are projected into the LiDAR-image. Bottom: an example of the occlusion filtering effect. Color codes distance from close (blue) to far point (red).

are available in [217].

Both the z-buffer and the occlusion filter were implemented in CUDA, allowing for realtime execution of the whole system.

6.2.2 Network Architecture

PWC-Net [17] was used as baseline, and we then made some changes to its architecture. We chose this network because PWC-Net has been designed to predict the optical flow between a pair of images, *i.e.,* to find matches between them. Starting from a rough camera localization estimate, our insight is to exploit the correlation layer of PWC-Net and its ability to match features from different points of view to regress the correct 6-DoF camera pose.

We applied the following changes to the original architecture.

- First, as our inputs are a depth and an RGB image (instead of two RGB images),

we decoupled the feature pyramid extractors by removing the weights sharing.

- Then, as we aim to perform pose regression, we removed the up-sampling layers, attaching the fully connected layers just after the first cost volume layer.

Regarding the regression part, we added one fully connected layer with 512 neurons before the first optical flow estimation layer (conv6_4 in PWC-Net), followed by two branches for handling rotations and translations. Each branch is composed of two stacked fully connected layers, the first with 256 neurons while the second with 3 or 4 neurons, for translation and rotation respectively.

Given an input pair composed of an RGB image $\mathcal{I}$ and a LiDAR-image $\mathcal{D}$, we used the loss function in Equation (6.3), where $\mathcal{L}_t(\mathcal{I}, \mathcal{D})$ is the translation loss and $\mathcal{L}_q(\mathcal{I}, \mathcal{D})$ is the rotation loss.

$$\mathcal{L}(\mathcal{I}, \mathcal{D}) = \mathcal{L}_t(\mathcal{I}, \mathcal{D}) + \mathcal{L}_q(\mathcal{I}, \mathcal{D}) \tag{6.3}$$

For the translation we used a smooth_{L1} loss [96]. Regarding the rotation loss, even though many approaches in the literature use the Euclidean distance, this distance does not provide a significant measure to describe the difference between two orientations. Instead, we used the angular distance between quaternions, defined as follows:

$$\mathcal{L}_q(\mathcal{I}, \mathcal{D}) = D_a(q * inv(\tilde{q})) \tag{6.4}$$

$$D_a(m) = atan2(\sqrt{b_m^2 + c_m^2 + d_m^2}, |a_m|) \tag{6.5}$$

Here, q is the ground truth rotation quaternion, $\tilde{q}$ represents the predicted normalized quaternion, inv is the inverse operation for quaternions, $\{a_m, b_m, c_m, d_m\}$ are the components of a quaternion m and $*$ is the multiplicative operation of two quaternions.

In order to use Equation (6.4) as a loss function, we need to ensure that it is differentiable for every possible output of the network. Recalling that $atan2(y, x)$ is not differentiable for $y = 0 \wedge x \leq 0$, and the fact that $\tilde{q}$ is a unit quaternion ($\|q\|_2 = 1$), we can easily verify that Equation (6.5) is differentiable in S^3.

6.2.3 Iterative Refinement

When the initial pose strongly deviates with respect to the camera frame, the map projection produces a LiDAR-image that shares just a few correspondences with the camera image. In this case, the camera pose prediction task is hard, because the CNN lacks the

required information to compare the two points of view. It is therefore quite likely that the predicted camera pose is not accurate enough. Taking inspiration from [216], we propose an iterative refinement approach. In particular, we trained different CNNs by considering descending error ranges for both the translation and rotation components of the initial pose. Once a LiDAR-image is obtained for a given camera pose, both the camera and the LiDAR-image are processed, starting from the CNN that has been trained with the largest error range. Then, a new projection of the map points is performed, and the process is repeated using a CNN trained with a reduced error range. Repeating this operation n times is possible to improve the accuracy of the final localization. The improvement is achieved thanks to the increasing overlap between the scene observed from the camera and the scene projected in the n^{th} LiDAR-image.

6.2.4 Training Details

We implemented CMRNet using the PyTorch library [218], and a modified version of the official PWC-Net implementation, as explained in Section 6.2.2. Regarding the activation function, we used a leaky ReLU with a negative slope of 0.1 as non-linearity. Finally, CMRNet was trained from scratch for 300 epochs using the ADAM optimizer with default parameters, weight decay of $5 * 10^{-6}$, and a batch size of 24 on a single NVidia GTX 1080ti. The initial learning rate was set to 0.0001 and halved after epochs 20, 50 and 70.

6.3 Experimental Evaluation

This section describes the evaluation procedure we adopted to validate CMRNet, including the used dataset, the assessed system components, the iterative refinements and finally the generalization capabilities.

We wish to emphasize that, in order to assess the performance of CMRNet itself, in all the performed experiments each input was processed independently, *i.e.,* without any tracking or temporal integration strategy.

6.3.1 Dataset

We tested the localization accuracy of our method on the KITTI dataset and on the RobotCar dataset.

For the KITTI dataset, we used the sequences 00, 03, and from 05 to 09 of the odometry sequences (for a total of 16 238 frames). We employed a leave-one-out cross validation approach, *i.e.,* for each sequence we used the other six sequences as training, and we tested the localization performance on the left-out sequence. Note that every sequence is spatially separated from the others thus the network is always tested in scenes never seen during the training phase. Since the accuracy of the provided GPS-RTK ground truth is not sufficient for our task (the resulting map is not aligned nearby loop closures), we used a LiDAR-based SLAM system to obtain consistent trajectories. The resulting poses are used to generate a down-sampled map with a resolution of 0.1 m. This choice is the result of our expectations on the format of HD-maps that will be soon available from map providers [206].

Since the images from the KITTI dataset have different sizes (varying from 1224x370 to 1242x376), we padded all images to 1280x384, in order to match the CNN architecture requirement, *i.e.,* width and height multiple of 64. Note that we first projected the map points into the LiDAR-image and then we padded both RGB and LiDAR-image, in order not to modify the camera projection parameters.

Concerning the RobotCar dataset, we considered a total of 42 runs, in particular we used the same runs of [219] except that we removed the three runs recorded during night. For each run, an image is stored every five meters of the vehicle's movement, only for the central camera (for a total of 51 927 frames). The RobotCar dataset contains different runs obtained by traversing the same path multiple times over a year, thus including changes in scene structure, lights and season. In this case, we geographically divided the path into different non overlapping regions, using four regions for validating and the rest for training. In particular, we used the same regions defined in [219]. Differently from the KITTI dataset, that provides LiDAR data from a Velodyne with 64 channels, the RobotCar dataset provided only single-channel SICK sensors. Therefore, the resulting maps contain fewer points than KITTI maps, and no down-sampling preprocess is needed. The images from the RobotCar dataset have shape of 960x1280, therefore the CNN requirement is already satisfied, and no padding is needed.

To simulate a noisy initial pose estimate H_{init}, we applied, independently for each input, a random translation and rotation to the ground truth camera pose. In particular, for each component, we added a uniformly distributed noise in the range of [-2 m, +2 m] for the translation and $[-10°, +10°]$ for the rotation.

Finally, we applied the following data augmentation scheme: first, we randomly changed the image brightness, contrast and saturation (all in the range [0.9, 1.1]). Then we randomly mirrored the image horizontally, and last we applied a random image rotation in the range $[-5°, +5°]$ along the optical axis. The 3D point cloud was transformed accordingly.

Both data augmentation and the selection of H_{init} take place at run-time, leading to different LiDAR-images for the same RGB image through the epochs.

6.3.2 System Components Evaluation

We evaluated the performances of CMRNet by assessing the localization accuracy, varying different sub-components of the overall system. In order to reduce the computational effort, we evaluated these tests only on the sequence 00 of the KITTI dataset (using the other sequences of the KITTI dataset as training). Moreover, the noise added to the poses on the validation set was kept fixed in all the experiments of this section, allowing for a fair comparison of the performances. Among the different sub-components of the system, the most significative are shown in Table 6.1, and derive from the following operational workflow.

First, we evaluated the best CNN to be used as backbone, comparing the performances of state-of-the-art approaches, namely PWC-Net, ResNet18 and RegNet [3, 17, 216]. According to the performed experiments, PWC-Net maintained a remarkable superiority with respect to RegNet and ResNet18 and therefore was chosen as a starting point for further evaluation.

Thereafter, we estimated the effects in modifying both inputs, *i.e.,* camera images and LiDAR-images. In particular, we added a random image mirroring and experimented different parameter values influencing the effect of the occlusion filtering presented in Section 6.2.1, *i.e.,* size K and threshold Th.

At last, the effectiveness of the rotation loss proposed in Section 6.2.2 was evaluated with respect to the commonly used L_1 loss. The proposed loss function achieved a relative decrease of rotation error of approximately 35%.

6.3.3 Iterative Refinement

In order to improve the localization accuracy of our system, we tested the iterative approach explained in Section 6.2.3. In particular, we trained three instances of CMRNet

Table 6.1: Parameter Estimation

	Occlusion				**Error**	
Backbone	*K*	*Th*	Mirroring	Rot. Loss	Transl.	Rot.
RegNet	-	-	✗	D_a	0.64 m	1.67°
ResNet18	-	-	✗	D_a	0.60 m	1.59°
PWC-Net	11	3.9999	✗	D_a	0.52 m	1.50°
PWC-Net	13	3.9999	✗	D_a	0.51 m	1.43°
PWC-Net	5	3.0	✗	D_a	0.47 m	1.45°
PWC-Net	5	3.0	✓	D_a	**0.46 m**	**1.36°**
PWC-Net	5	3.0	✓	L_1	0.46 m	2.07°

Median localization accuracy varying different sub-components of the overall system. *K* and *Th* correspond to the occlusion filter parameters as described in Section 6.2.1.

varying the maximum error ranges of the initial camera poses. To assess the robustness of CMRNet, we repeated the localization process for 10 times using different noises on the initial position estimates. The averaged results are shown in Tables 6.2 and 6.3, for the KITTI and the RobotCar dataset respectively, together with the correspondent ranges used for training each network. While Table 6.2 shows the overall results on all the KITTI odometry sequences, in Table 6.6 we reported the results on each sequence separately. We believe that the better results obtained in the RobotCar dataset are due to the higher density of the points in the map (since no subsample operation was performed). In order to confirm these thoughts, we tested the localization performances of CMRNet on the sequence 00 of the KITTI dataset varying the density of the points in the LiDAR-map. The results reported in Table 6.5 show a decrease of the accuracy when the density of the points is reduced. Although this is a preliminary test, and can not prove our statement, the results support our supposition.

Moreover, in order to compare the localization performances with the state-of-the-art monocular localization in LiDAR maps [213], we calculated mean and standard deviation for both rotation and translation components over 10 runs on the sequence 00 of the KITTI odometry dataset. Our approach shows comparable values for the translation component (0.33 ± 0.22 m compared to 0.30 ± 0.11 m), with a lower rotation errors ($1.07 \pm 0.77°$ compared to $1.65 \pm 0.91°$). Nevertheless, it is worth to note that our approach still does not take advantage of any pose tracking procedure nor multi-frame

Table 6.2: Iterative Pose Refinement — KITTI

	Initial Error Range		Localization Error	
	Transl. [m]	Rot. [deg]	Transl. [m]	Rot. [deg]
Iteration 1	[-2, +2]	[−10, +10]	0.54	1.62
Iteration 2	[-1, +1]	[−2, +2]	0.35	1.29
Iteration 3	[-0.6, +0.6]	[−2, +2]	**0.32**	**1.25**

Median localization error at each step of the iterative refinement on all the sequences of the KITTI odometry dataset (sequences 00, 03, and 05 to 09). The results are averaged over 10 different initial pose estimation for each frame.

Table 6.3: Iterative Pose Refinement — RobotCar

	Initial Error Range		Localization Error	
	Transl. [m]	Rot. [deg]	Transl. [m]	Rot. [deg]
Iteration 1	[-2, +2]	[−10, +10]	0.28	0.64
Iteration 2	[-1, +1]	[−2, +2]	0.15	0.38
Iteration 3	[-0.3, +0.3]	[−1, +1]	**0.11**	**0.25**

Median localization error at each step of the iterative refinement on the RobotCar dataset averaged over 10 runs.

analysis.

Some qualitative examples of the localization capabilities of CMRNet with the latter iteration scheme are depicted in Figures 6.3 and 6.4, on the KITTI and RobotCar dataset respectively.

In Figure 6.5 we illustrate the probability density functions (PDF) of the localization error on the KITTI dataset, decomposed into the six components of the pose, for the three iterations of the iterative refinement. It can be noted that the PDF of even the first network iteration approximates a Gaussian distribution and following iterations further decrease the variance of the distributions.

Analysis of the runtime performances on both datasets using this configuration are shown in Table 6.4. Since the images in the RobotCar dataset are bigger than the ones in KITTI, the inference time for the RobotCar dataset is higher, but CMRNet still maintains realtime capabilities.

Table 6.4: Runtime Performances

	Z-Buffer	Occlusion Filter	CMRNet	**Total**
KITTI	8.9 ms	1.4 ms	4.6 ms	**14.7 ms** (∼68 Hz)
RobotCar	8.0 ms	1.8 ms	8.6 ms	**18.4 ms** (∼55 Hz)

In the table, an analysis of the time performances of the system steps for a single iteration, on both datasets, *i.e.,* 44.1 ms and 55.2 ms for the 3-stages iterative refinement. All the code was developed in CUDA, achieving 68 fps runtime performances on the KITTI dataset and 55 fps on RobotCar. CPU-GPU transfer time was not here considered.

Table 6.5: Points Density Test

	Density 10 cm		Density 15 cm		Density 20 cm	
	Tr. [m]	Rot. [deg]	Tr. [m]	Rot. [deg]	Tr. [m]	Rot. [deg]
Iteration 1	0.48	1.45	0.61	1.65	0.68	1.68
Iteration 2	0.31	1.15	0.39	1.33	0.48	1.42
Iteration 3	**0.28**	**1.11**	0.33	1.21	0.41	1.30

This table shows the performances of CMRNet on the sequence 00 of the KITTI dataset varying the density of the points in the LiDAR-map. Please note that the initial error ranges for the 3 iterations are the same used in Table 6.2.

Table 6.6: Iterative Pose Refinement — KITTI

	Seq 00		Seq 03		Seq 05		Seq 06		Seq 07		Seq 08		Seq 09	
	Tr.	Rot.	Tr.	Rot.	Tr.	Rot.	Tr.	Rot.	Tr.	Rot.	Tr.	Rot.	Tr.	Rot.
Iteration 1	0.48	1.45	0.71	1.61	0.55	1.66	0.62	1.91	0.42	1.35	0.56	1.66	0.62	1.96
Iteration 2	0.31	1.15	0.48	**1.46**	0.39	1.54	**0.48**	1.29	0.26	0.98	0.34	**1.33**	0.39	1.37
Iteration 3	**0.28**	**1.11**	**0.42**	1.50	**0.35**	**1.54**	0.51	**1.24**	**0.25**	**0.86**	**0.31**	**1.33**	**0.34**	**1.31**

Median localization error at each step of the iterative refinement on the KITTI sequences. The initial errors ranges are [±2 m, ± 10°] for the first iteration, [±1 m, ± 2°] for the second and [±0.6 m, ± 2°] for the third

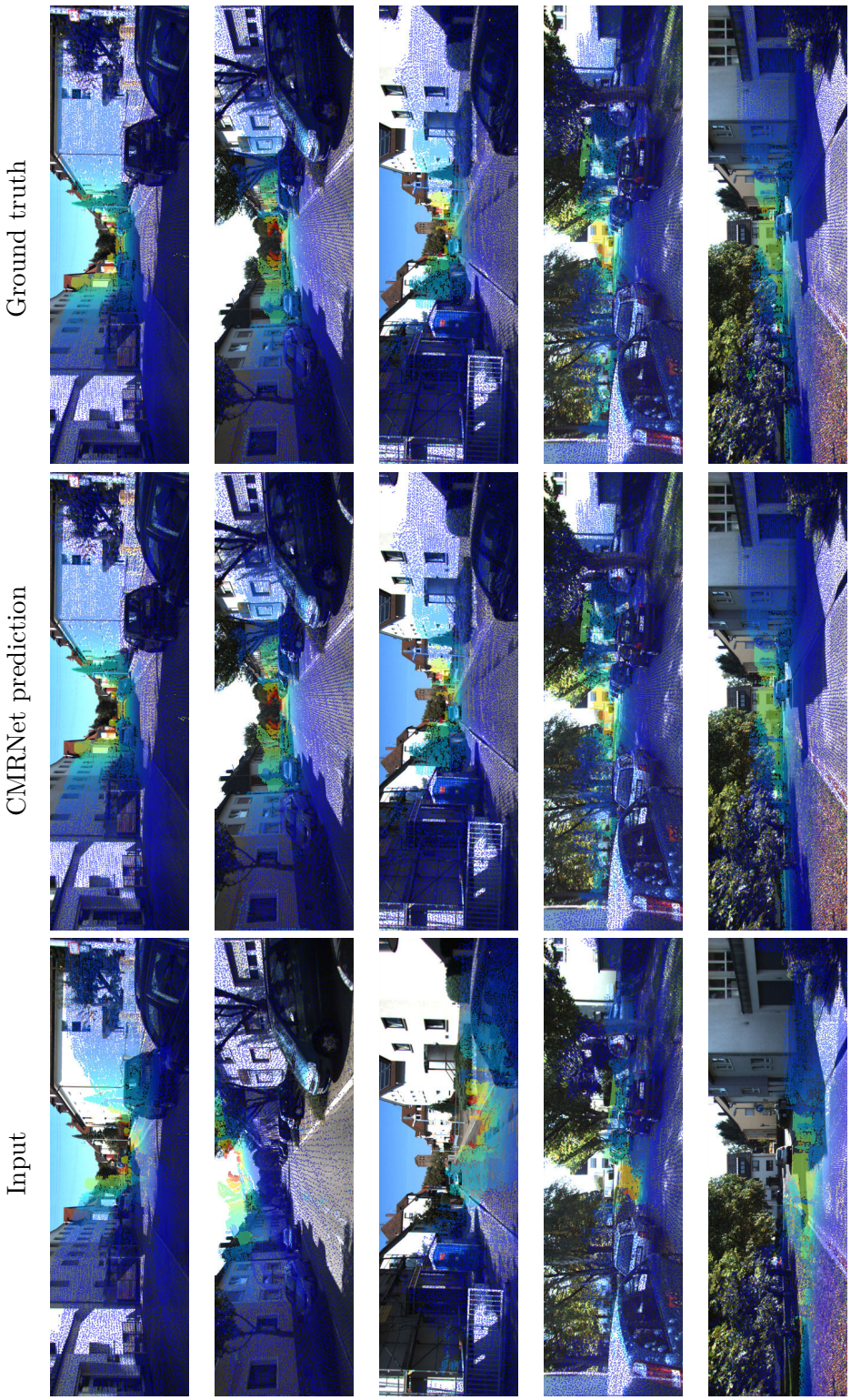


Figure 6.3: Five examples of the localization results on the KITTI dataset. From left to right: Input LiDAR-image, CMRNet result after the third iteration, ground truth. All LiDAR-images are overlaid with the respective RGB image for visualization purpose.

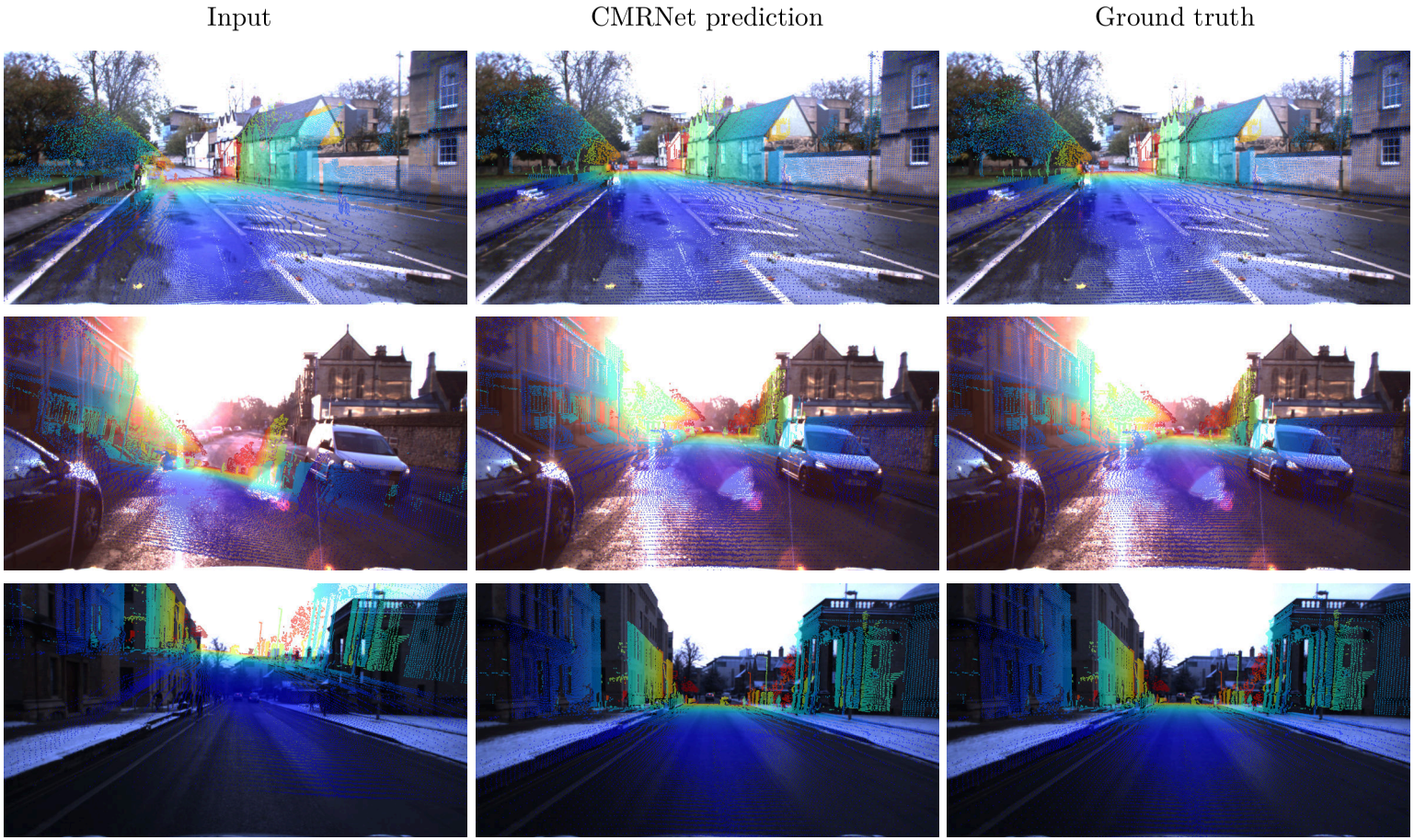
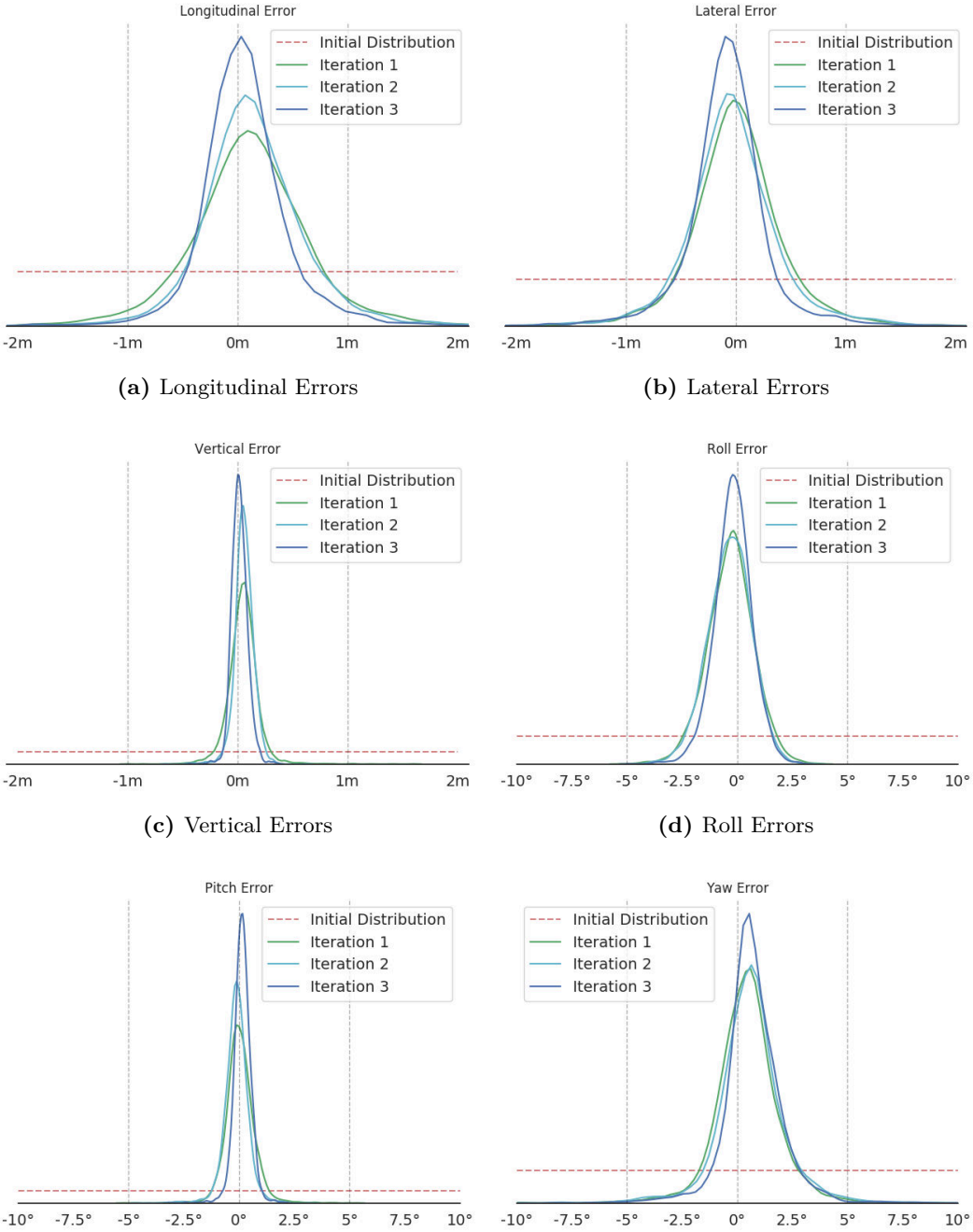


Figure 6.4: Three examples of the localization results on the RobotCar dataset in challenging situations (rain, direct sunlight, snow). From left to right: Input LiDAR-image, CMRNet result after the third iteration, ground truth. All LiDAR-images are overlaid with the respective RGB image for visualization purpose.



(a) Longitudinal Errors

(b) Lateral Errors

(c) Vertical Errors

(d) Roll Errors

(e) Pitch Errors

(f) Yaw Errors

Figure 6.5: Iterative refinement error distributions on the sequence 00 of the KITTI dataset. A PDF has been fitted (using Gaussian kernel density estimation) on the network error outcome for each iteration step and each component. The dashed red lines are the theoretic PDFs of the initial H_{init} errors.

6.3.4 Generalization Capabilities

In order to assess the generalization effectiveness of our approach, we evaluated its localization performance using a 3D LiDAR-map generated on a different day with respect to the camera images, yet still of the same environment. This allows us to have a completely different arrangement of parked cars and therefore to stress the localization capabilities.

Unfortunately, there is only a short overlap between the sequences of the KITTI odometry dataset (approx. 200 frames), consisting of a small stretch of roads in common between sequences "00" and "07". Even though we cannot completely rely on the results of this limited set of frames, CMRNet achieved 0.57 m and 0.9° median localization accuracy on this test.

Indeed, it is worth to notice that the network was trained with maps representing the same exact scene of the respective images, *i.e.,* with cars parked in the same parking spots, and thus cannot learn to ignore cluttering scene elements.

The RobotCar dataset, on the other hand, would in principle be well suited for testing the localization errors of CMRNet on maps built on a different day than the query images. However, the precision of the positions provided with the dataset "*varied significantly during the course of data collection*", and the authors suggest to not use them as "*ground truth for benchmarking localisation and mapping algorithms, especially if using multiple traversals spaced over many months*" [181]. Moreover, due to the single-channel LiDAR devices provided with the dataset, the alignment of maps taken during different days through SLAM-like techniques is not a trivial task. Therefore, we were unable to evaluate CMRNet within maps generated on a different day on the RobotCar dataset.

Thereafter, we evaluated the localization performances of our approach in illumination condition different from the ones used in the training phase. In particular, we tested CMRNet in a run of the RobotCar dataset recorded during night-time (run "2014-12-16-18-44-24"). Even though the network was trained only with images recorded during day-time, CMRNet achieved 0.34 m and 0.91° median localization accuracy in this test. Some qualitative examples of this test are depicted in Figure 6.6. The great localization performances achieved by our approach during night-time demonstrate the robustness of CMRNet to illumination variations.

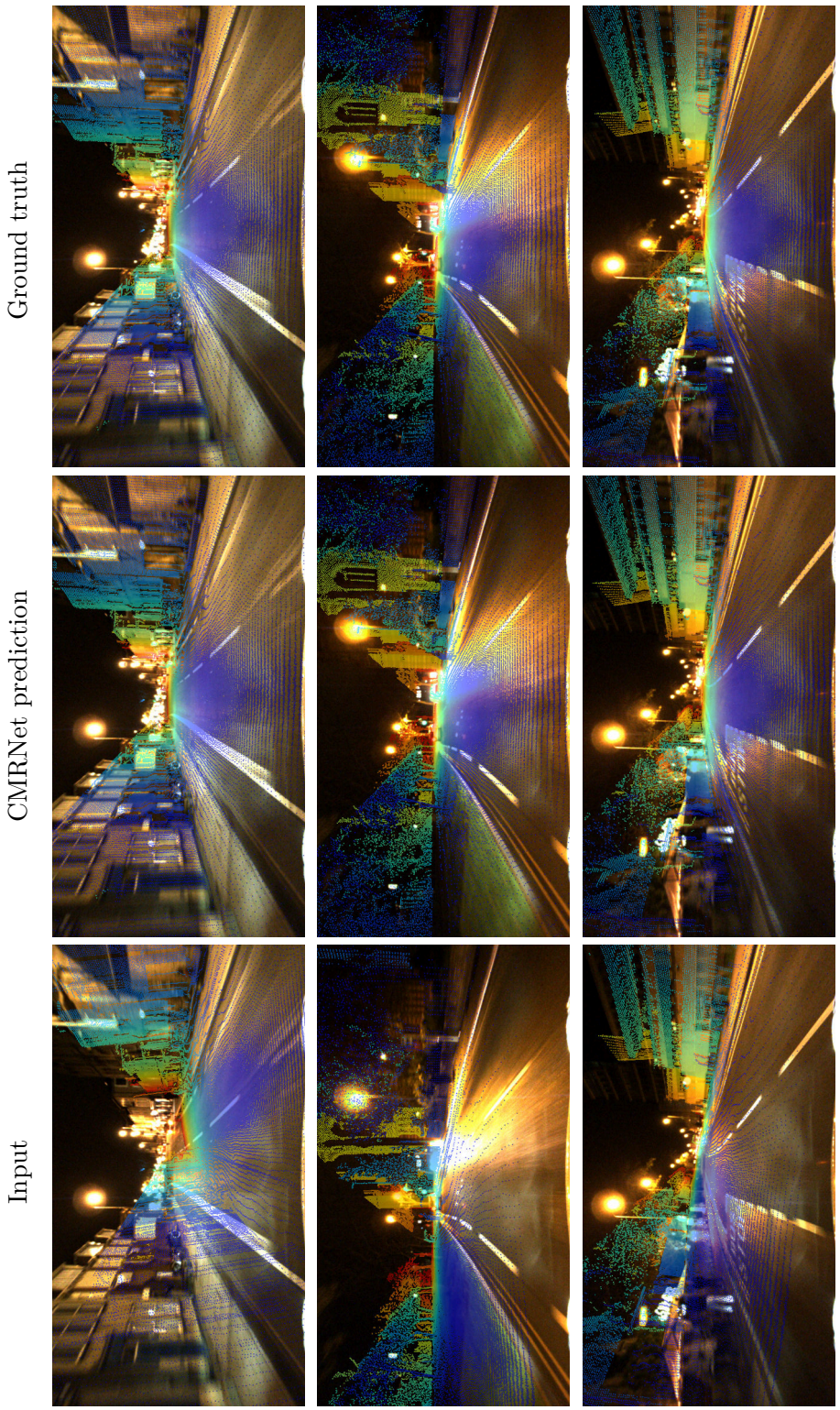


Figure 6.6: Three examples of the localization results on the RobotCar dataset during nighttime. Please note that CMRNet was trained only with images recorded during daytime. From left to right: Input LiDAR-image, CMRNet result after the third iteration, ground truth. All LiDAR-images are overlaid with the respective RGB image for visualization purpose.

6.4 Conclusions

In this chapter, we described CMRNet, a CNN-based approach for camera to LiDAR-map registration. The runtime performances of the proposed approach allow multiple specialized instances of CMRNet to be stacked to improve the final camera localization, yet preserving realtime requirements. The results have shown that our proposal is able to localize the camera with a median less than 0.32 m and 1.25° on the KITTI odometry datasets and 0.11 m and 0.25° on the Oxford RobotCar dataset. Therefore, CMRNet achieved "lane-level" localization accuracies in urban environments even in challenging conditions (*e.g.,* night, snow, direct sunlight). Moreover, since the error distributions (depicted in Figure 6.5) reveal a similarity with Gaussian distributions, we expect to be able to benefit from standard filtering techniques aimed to probabilistically tackle the uncertainties over time. Since the proposed method does not learn the map, but learns how to perform the registration, it is suitable for being used with large-scale HD-maps. Moreover, we proposed a new angular loss for the rotation components that achieved a relative rotation errors reduction of approximately 35%.

Chapter 7

Visual place recognition within HD-maps

So far in this thesis, we have addressed the local localization task in challenging environments. In particular, in Chapters 3 to 5 we leveraged topological maps (the OSM service) to localize a vehicle in highway-like and urban environments by exploiting high-level features such as ego-lane, road intersections, and buildings. In Chapter 6, instead, we proposed a realtime CNN-based approach to localize a single camera image within LiDAR-maps. Recalling that local localization techniques require an initial position estimate to be known, all the approaches proposed so far require a rough position as input. Usually, this rough position can be obtained using GNSSs, which provides a global localization. However, the reliability of GNSSs is often inadequate for autonomous driving applications, especially in urban environments and indoor areas (*e.g.,* indoor parking). Signals from satellites might be blocked by buildings or bridges, for example, leading to no position estimates at all. Being able to provide a "global" localization, when GNSSs are not available, require further scene understanding techniques.

A common way to solve the latter problem is to use "visual place recognition" approaches [220–223]. These approaches aim to localize a query image by comparing it with a prerecorded database of geo-referenced images. If the most similar image retrieved by the approach depicts the same place of the query image, the retrieval is considered correct, and the position of the retrieved image serves as a rough global localization. The authors in [61], instead, proposed to utilize visual odometry and the road graph from the OSM service to globally localize a vehicle at city-scale. An extension of the latter approach that also utilizes sun direction, presence of an intersection, road type (*highway* or *non-highway*), and speed limits was proposed in [224]. Both approaches, however, require an image sequence of about 20 seconds of driving before estimating the vehicle's localization, and therefore they are not suitable for fully autonomous vehicles.

Recent machine learning approaches, mainly based on CNNs or random forests, tackle the localization problem in a single step by regressing the robot's position using only a single RGB image [21, 23, 209], without any map or prior information about its position. However, they have important limitations. First, they require a huge amount of images of the environment to train the model, and second, they achieved good performances only in small environments (*e.g.,* a room or a small outdoor area of few tens of meters). Moreover, these approaches can not be used to obtain a localization in scenes that were not represented in the training set, since the model learns to perform localization only with respect to the specific places used during the training phase. Therefore, for new locations, the acquisition of a new dataset, together with a re-training of the model are required. This characteristic makes these methods not suitable for autonomous road vehicles.

As we stated in Chapter 6, our research is now focused on localization with respect to HD-maps. To effectively exploit the forthcoming market of these maps, however, we need a reliable method to perform global localization within HD-maps in GNSSs-denied environments. Existing approaches for global localization within LiDAR-maps only exploit LiDAR sensors onboard the vehicle, and perform the localization by means of point cloud registration algorithms [203, 225, 226] or LiDAR place recognition techniques [219, 227, 228].

How to perform global **visual** localization with respect to LiDAR-maps when GNSSs are not available is still an open question, even though the path toward high definition 3D maps, produced mainly from LiDAR sensors, is clear. In this chapter we propose a novel DNN-based approach to the global visual localization problem, *i.e.,* to localize an RGB image in a previously built geometric 3D map of an urban area. Specifically, we propose to jointly train two DNNs, one for the images and the other for the point clouds, in order to generate a shared embedding space for the two types of data. This embedding space allows us to perform place recognition with heterogeneous sensors. In fact, an image query with respect to a point cloud database is just one of the possibilities. Besides allowing localization on a geometric map using visual information, our proposal does not need to be trained for every new environment. Instead, the neural networks are trained with samples from many different areas. In this way, the networks learn to associate similar descriptors to images and point clouds representing the same place. Therefore, our approach can associate an image, *i.e.,* the query, to the corresponding

point cloud, among those that constitute the map of the area, *i.e.,* the database. An extensive experimental activity is presented to assess the effectiveness of the approach with respect to different learning methods, network architectures, and loss functions. All the evaluations have been performed using the Oxford RobotCar Dataset, which encompasses a wide range of weather and light conditions. To the best of our knowledge, this is the first approach for global visual localization within city-scale 3D-LiDAR map.

7.1 Related Work

Historically, visual place recognition represents a relevant problem for the computer vision community. The aim of this task is to decide whether and where a place, observed in an image, has already been observed within a set of geo-referenced images. This is relevant in mobile robotics, as it may serve as a starting point for local localization systems, or to detect loop closures in SLAM systems [229, 230].

Traditional methods involve an offline pre-computation of image descriptors for the images of a database of images, which are then compared online against the descriptors of a query image, to retrieve the most similar place. Methods to perform place recognition were based on approaches such as bag of words [220], which exploit handcrafted-features (*e.g.,* SIFT and SURF [43, 44]), visual vocabulary [221], and query expansion [222]. More recently, methods that exploit deep learning techniques have been proposed. As an example, the authors in [223] exploited a CNN-based feature extractor and proposed a specific pooling layer, named NetVlad, designed for place recognition, which was inspired by the Vector of Locally Aggregated Descriptors (VLAD) [231].

The definition of discriminative descriptors represents a fundamental step of the place recognition task. Within machine learning methodologies, this task is usually performed using *metric learning* techniques. The goal of metric learning is to find an appropriate distance (or similarity) function for a specific task, by inferring such function from the data. In order to learn non-linear distance functions, a common way is to instead learn a non-linear mapping from the inputs to a feature space, and then compute the similarity through a "simple" distance function (such as Euclidean or cosine). The resulting feature space is called embedding space, and the descriptors take the name of Embedding Vectors (EVs). We also want such embedding space to be low-dimensional, so to perform fast comparison between samples. In recent years, a common trend is to learn this non-linear mapping using DNNs, in this case we talk about *deep metric learning.*

A possible method to train such DNNs, named *contrastive*, provides sample pairs to a model by considering two cases: either the two samples represent the same concept or distinct ones [232]. This training method tries to minimize the distance between EVs of the same class while penalizing small distances between EVs of different classes. Another similar technique consists in comparing *triplets* of samples, which are organized such that a given input is associated with a positive sample and a negative one. In particular, positive samples represent the same concept of the input, while the negative samples belong to a different one. Different approaches extended the triplet embedding method by considering multiple negative samples at once instead of a single one [233, 234].

Although place recognition is a task traditionally reserved to vision-based systems, recently a DNN-based approach that performs place recognition on LiDAR-maps has been proposed [219]. Their insight was to exploit the strong independence of LiDAR data with respect to lighting conditions. This method, named PointNetVLAD, matches an input point cloud, *e.g.,* acquired from a vehicle, to another one among those contained in a database representing a large urban area. It can match (the point cloud of) a LiDAR scan to a location in a LiDAR-map of a large urban area. PointNetVLAD uses PointNet [110] as feature extractor, and NetVlad [223] to compute the descriptors.

To the best of our knowledge, the only work in the literature that treats the problem of image and LiDAR-map descriptors matching, using a metric learning approach, has been proposed in [235]. In their work a neural network, named *2D3D-MatchNet*, performs descriptors extraction from 2D image patches and 3D LiDAR patches. The objective is to perform the metric localization of a camera through 2D to 3D patch matching. Although a neural network is used to compute the descriptors, 2D key-points are detected from images using the detector in the SIFT software, while 3D key-points are extracted from the point clouds using Intrinsic Shape Signatures [236]. The key-points are then used to define the patches used for computing the descriptors, from images and LiDAR-maps. The localization is finally computed solving a Perspective-n-Point (PnP) problem. Their work is limited also in that they do not handle LiDAR-maps whose extension is larger than a few tens of meters, which is not global localization.

There are many other contributions in the literature, which match image descriptors to 3D maps, but they refer to maps produced by SfM approaches, where one or more image descriptors are attached to the 3D-reconstructed points. On the other hand, we need to handle descriptor-less point clouds, as naturally coming from LiDAR sensors.

Despite the results obtained on images, the field of neural networks for 3D data processing is not largely explored and DNNs for 3D LiDAR data have appeared only recently. Moreover, a neural network that can handle very large point clouds, such as those representing cities, still does not exist. Therefore, the selection of a model that can generate effective EVs from LiDAR maps is still an open problem. The first approaches that processed point clouds with DNNs [108, 109] first converted the cloud into a 3D occupancy grid, and then applied 3D CNNs. However, such 3D CNNs are computationally expensive, limiting the size of the input point clouds. In order to overcome this limitation, the authors in [110] proposed PointNet, a neural network that directly consumes the cloud as a list of unordered points. PointNet is one of the most influential work on 3D DNNs, that inspired many subsequent works. For example, PointNet++ [111] added sampling and grouping strategies to allow local features extraction, while *EdgeConv* [112] extracted local geometries by computing a neighborhood graph and using convolution operations on the edges of the graph. One of the innovations introduced by EdgeConv concerns the capability of extracting similar features for semantically similar structures of an object. Another way to overcome the limitations associated with 3D CNNs is to exploit hierarchical data structures. For instance, Kd-network [113] exploits kd-trees as the underlying data structure, while O-CNN [114] utilizes the octree representation (Section 2.2.2). A recent trend for tackling the 3D object detection task is to use a hybrid approach: the input point cloud is first divided into a voxel grid, and then a PointNet-like DNN extract a feature vector for each non-empty voxel, by processing all the points that fall in that voxel. The resulting sparse voxel grid is then processed using 3D and 2D convolutional networks. *VoxelNet* [103] was the pioneer of this kind of approaches, and achieved state-of-the-art performances on the KITTI 3D object detection benchmark [179]. The novelty and the great results achieved by VoxelNet inspired different subsequent works. For example, SECOND [104] used sparse and submanifold convolutions to reduce the execution time and improve inference accuracy, while PointPillars [237] used a 2D grid in the x-y plane instead of a 3D grid.

7.2 Proposed Approach

Differently from visual place recognition techniques, our approach matches a single RGB image to a database of known-pose point clouds, to retrieve the point cloud representing the same place of the query image. In order to compare images and point clouds, we

propose to learn a shared embedding space where data representing the same place live close to each other even though they have to be computed from different types of data. In particular, we propose two DNNs, one for the images and another for the point clouds, jointly trained to produce similar EVs, when the image and the point cloud come from the same place.

Our work was inspired by PointNetVLAD [219], which requires a LiDAR device onboard the vehicle, our approach instead only requires a camera onboard, which is the focus of this thesis.

More formally, given a query image $\mathcal{I}$ and a LiDAR-map $\mathcal{M}$, we can split the map into multiple overlapping sub-maps m_i. We formulate the global localization task as a metric learning problem: we want to find two mapping functions $f(\cdot)$ and $g(\cdot)$, implemented as two DNNs such that $d(f(\mathcal{I}), g(m_i)) < d(f(\mathcal{I}), g(m_j))$ when m_i represent the same place where $\mathcal{I}$ was taken, and m_j does not. $d(\cdot)$ is a distance function, such as the euclidean distance. The domain of $f(\cdot)$ is the image space ($\mathbb{R}^{HxWx3}$, Height x Width x 3 channels: RGB), while the domain of $g(\cdot)$ is the point cloud space ($\mathbb{R}^{Nx3}$, N points x 3 coordinates: XYZ). The output of both functions is an EV of fixed length K; therefore the co-domain is $\mathbb{R}^K$. Once we have found the mappings between the input spaces and the embedding space, to compute a global localization, *i.e.,* to determine the most similar place, we have to compare the EV of the query image with the EVs of all the sub-maps $m_i \in \mathcal{M}$.

7.2.1 2D Network Architecture

The network that computes the EVs from images is composed of two parts. First, a CNN extracts local features from the input image; then those features are aggregated to provide a fixed-length EV. Concerning the CNN, we considered some of the most relevant architectures for the image classification task. In particular, we tested the following networks: VGG-16 [238] and ResNet-18 [3]. Since we are not interested in the classification task, we removed the fully connected layers from the latter architectures, cropping them after the last convolutional layer. For the aggregation step, we tested the NetVlad layer [223], that was specifically designed for the place recognition task.

7.2.2 3D Network Architecture

Similarly to the 2D CNN architecture, for the 3D network we adopted the same approach: a DNN-based features extractor followed by a features' aggregation layer. The best data

structures and architectures that allow the extraction of discriminative descriptors from 3D point clouds are not evident in the literature, therefore a comparison between different architectures was needed. The first 3D feature extractor considered was PointNet [110], since it was the first 3D DNN approach to directly process point clouds as a list of points. We also considered one interesting extension of PointNet, named EdgeConv [112]. The latter DNNs require point clouds composed of a fixed number of points, thus a down-sampling step of the input is necessary. The last option we considered is the feature extraction layer of SECOND [104]. Since we are not interested in classification or segmentation tasks, we modified the latter networks as follows:

- PointNet: we cropped the network before the MaxPooling layer;
- EdgeConv: we cropped the segmentation network after the concatenation layer;
- SECOND: we replaced the RPN with an Atrous Spatial Pyramid Pooling (ASPP) layer [11]

Finally, for the aggregation step, again we used the NetVlad layer.

These models accommodate different representations of the input data (*e.g.,* unordered lists, graphs, voxel grids), thus we think that providing an evaluation of the existing approaches is an important step forward for this type of research.

7.2.3 Learning Methods

We propose two different learning methods to create a shared embedding space between 2D and 3D EVs: *teacher/student*, and *combined*.

Teacher/Student Training

Given a 2D and a 3D neural network, the teacher/student method first trains one of the network (the teacher) to create an effective embedding space for a particular task. In a second step, this pre-trained network will be used to help the second one (the student) to also generate a similar embedding space, *i.e.,* similar EVs for similar concepts. For instance, initially a CNN model learns to perform place recognition with images, then a 3D DNN tries to emulate the CNN descriptors. In this way, the network creates the embedding space where EVs are defined and the student also tries to generate that space from a different kind of data.

In this work, we first train the 2D network with a triplet loss function [239]: given a triplet (I_i^a, I_i^p, I_i^n) composed of an anchor image I_i^a, an image depicting the same place I_i^p (*positive*) and an image of a different place I_i^n (*negative*), the loss function is defined as:

$$\mathcal{L}_{trp}^{2D\text{-to-}2D} = \sum_i [d(f(I_i^a), f(I_i^p)) - d(f(I_i^a), f(I_i^n)) + m]_+ \tag{7.1}$$

where $d(\cdot)$ is a distance function, $f(\cdot)$ is the 2D network we want to train, $[\cdot]_+$ means $Max(0, [\cdot])$, and m is a margin that is enforced between positive and negative distances.

Once the 2D network (teacher) has been trained, the 3D network (student) is trained to mimic the output of the teacher, so to obtain a Joint Embedding (JE). In this case, given a pair (I_i, m_i) composed of an image I_i and a point cloud m_i captured at the same time, the loss function is:

$$\mathcal{L}^{JE} = \sum_i d(f(I_i), g(m_i)) \tag{7.2}$$

Please note that, during this step, only the student network $g(\cdot)$ is trained, while the teacher $f(\cdot)$ is kept fixed.

Combined Training

Alternatively, we propose a combined approach that simultaneously trains both the 2D and the 3D neural networks, in order to produce the same embedding space. In this case, the loss function proposed to jointly train both networks is composed of different components.

Same-Modality metric learning. The first components of the proposed combined loss are aimed at producing effective EVs for place recognition within the same modality, *i.e.,* the same type of sensor data (query image with respect to image database and query point cloud with respect to point cloud database). The same-modality loss is defined as follows:

$$\mathcal{L}_{trp}^{SM} = \mathcal{L}_{trp}^{2D\text{-to-}2D} + \mathcal{L}_{trp}^{3D\text{-to-}3D} \tag{7.3}$$

Here, $\mathcal{L}_{trp}^{2D\text{-to-}2D}$ is the 2D-to-2D triplet loss defined in Equation (7.1), while the 3D-to-3D loss $\mathcal{L}_{trp}^{3D\text{-to-}3D}$ can be derived similarly.

Cross-Modality metric learning. In order to learn 2D and 3D EVs that live in the same

embedding space, we extend the triplet loss to perform cross-modality metric learning:

$$\mathcal{L}_{trp}^{2D\text{-to-}3D} = \sum_i [d(f(I_i^a), g(m_i^p)) - d(f(I_i^a), g(m_i^n)) + m]_+ \tag{7.4}$$

$$\mathcal{L}_{trp}^{3D\text{-to-}2D} = \sum_i [d(g(m_i^a), f(I_i^p)) - d(g(m_i^a), f(I_i^n)) + m]_+ \tag{7.5}$$

$$\mathcal{L}_{trp}^{CM} = \mathcal{L}_{trp}^{2D\text{-to-}3D} + \mathcal{L}_{trp}^{3D\text{-to-}2D} \tag{7.6}$$

An example of the $\mathcal{L}_{trp}^{2D\text{-to-}3D}$ loss computed on a triplet (I_i^a, m_i^p, m_i^n) is depicted in Figure 7.1.

Joint Embedding loss. The last component of the proposed loss tries to minimize the distance between 2D and 3D EVs recorded at the same time, *i.e.,* we want the EV of a point cloud to be as close as possible to the EV of the corresponding image. To achieve this aim we used the joint embedding loss defined in Equation (7.2); however, in this case both networks $f(\cdot)$ and $g(\cdot)$ are trained.

Full combined loss. The final loss used to jointly train the 2D and 3D networks is a combination of the aforementioned components ($\lambda_1 = 0.1, \lambda_2 = 1, \lambda_3 = 1$):

$$\mathcal{L}_{total} = \lambda_1 \mathcal{L}_{trp}^{SM} + \lambda_2 \mathcal{L}_{trp}^{CM} + \lambda_3 \mathcal{L}^{JE} \tag{7.7}$$

7.2.4 Training Details

During the training phase, to mitigate overfitting, we use the following data augmentation scheme: for the images, we first apply a random color jitter (changing the saturation, brightness, hue and contrast). Then, we randomly rotate the image within a range of $[-5°, +5°]$; finally, we apply a random translation on both axes, with a maximum value of 10% of the size of the image on that axis. Regarding the point clouds, we applied a random rigid body transformation, with a maximum translation of 1.5 meters on all the axes, and a maximum rotation of 10° around the vertical axis and 2° around the lateral and longitudinal axes. We chose these values to accommodate slightly different points of view.

Moreover, we applied random horizontal mirroring to the images and, when an image is mirrored, the relative point cloud is also mirrored. This particular augmentation is applied before any other data augmentation.

For the implementation we used the PyTorch library [218]. The input images are

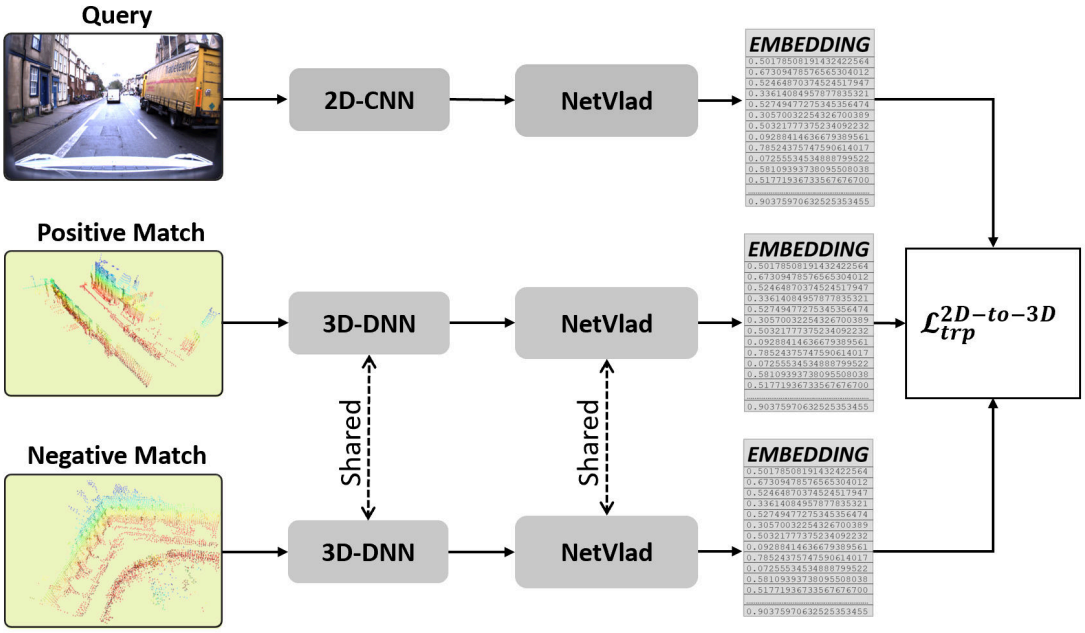


Figure 7.1: During the training phase, the "triplet" technique considers a positive and a negative sample with respect to a query. Please note that the weights of the 3D-DNN and the associated NetVlad layers are shared between the point cloud samples, meaning that we use the same network.

undistorted and resized to 320x240. The batch size is constructed by randomly selecting N places and randomly picking two samples of each of the selected places, so to have N positive pairs within the batch. The negative samples are selected randomly (one for each positive pair) within the batch.

7.2.5 Inference

Once the DNN models have been trained, the inference of the place recognition is pursued as follows. First, the 3D LiDAR-map of the environment is split into overlapping sub-maps and the 3D DNN is used to generate the corresponding EVs. These EVs are then organized in a specific data structure in order to allow a subsequent fast nearest neighbor querying, in particular we used a KD-tree approach. Lastly, the localization is performed by comparing the EV of the query image, generated by the 2D CNN, with respect to the database. It would of course be possible to reverse the paradigm, *i.e.,* to produce a dataset of 2D image EVs and then perform queries using the EVs from 3D sub-maps or

LiDAR scans.

7.3 Experimental Evaluation

This section describes the experiments performed to evaluate the different architectures and approaches, including the dataset preprocessing task.

7.3.1 Dataset

In order to develop an approach that is robust to challenging environmental conditions (*e.g.,* scene structure, lights and seasonal changes), we used the Oxford RobotCar [181] dataset for both training and validation. The KITTI dataset, which we used for evaluating the approaches proposed in the previous chapters, is not suitable for the place recognition task, because the different sequences almost never geographically intersect each other. The RobotCar dataset, on the other hand, is composed of multiple runs obtained by traversing the same path multiple times over a year; therefore, it is well suited for the place recognition task. RobotCar contains geo-referenced LiDAR scans, images, and GPS data of urban areas. It includes a broad set of weather conditions, which allowed us to develop a robust approach. For instance, it is possible to test the place recognition performance by considering different light conditions for images, and structural changes for the scene, which are reflected in corresponding changes in the LiDAR-maps, *e.g.,* in the presence of roadworks. RobotCar has also been used for training and testing PointNetVLAD [219], which inspired our research.

7.3.2 Regions Subdivision and Sub-maps Creation

As in Chapter 6, we considered the same 45 runs used in [219], in this case we also included the night runs. For each run, an image is stored every five meters and the corresponding LiDAR sub-map is cropped from the whole map. For each 3D sub-map we considered a range of 50 meters on each axis. The training and validation sets have been created by performing a subdivision of the global path in different non-overlapping regions. The goal here was to make it possible to evaluate the performance in regions never provided to the neural network during the training phase. In particular, we used the same four regions that we used in Chapter 6 and defined in [219].

7.3.3 Evaluation Metrics

The evaluation of a place recognition approach is usually based on the *recall* measure. When the number of samples in the database increases, the difficulty of the operation increases as well. For this reason, metrics such as *recall@k* are used, in order to provide a fair evaluation. The term k represents the number of places that our system determines to be the most similar with respect to the input query. If at least one of the retrieved k elements corresponds to the location of the input query, then the retrieval is considered correct. We fixed k to the 1% of the samples contained in the database in order to ensure the invariance of the measure with respect to the database size. In our experiments, we considered two poses to belong to the same place if their distance is less than $20m$.

To test the performance of our approach we proceeded as follows. For every possible pair of distinct runs, again as in [219], we used all samples of the first run as database, and only the samples within the validation area of the second run as queries. In this way, we tested our method in places never seen during the training phase. Finally, we compute the mean of the recall@k metric over all pairs.

7.3.4 Results

We based on the networks proposed in [223] (VGG-16+NetVLAD), and on a modified version of [104] (SECOND+ASPP+NetVLAD) in order to extract EVs from both 2D and 3D data. We applied the teacher/student learning method with the smooth-L1 distance function [96], fixing the 2D architecture as teacher, since it represents the state-of-the-art in the place recognition task with images. In particular, we took a pre-trained version of the 2D backbone and performed fine-tuning on the Oxford RobotCar dataset.

The results of these experiments are provided in Table 7.1, where we show the performance with respect to all the possible query-database combinations. Even though our work is mainly focused on the 2D-to-3D modality, we also obtained comparable results in the 2D-to-2D place recognition modality, and even state-of-the-art performances for 3D-to-3D. Considering the novelty of the proposed approach, the obtained results are promising. An important aspect concerns the recall achieved when performing 3D queries on 3D database: we found that our approach, which exploits SECOND, outperforms PointNetVLAD [219] on the same task, as shown in Table 7.2. To perform a fair comparison we followed an evaluation scheme similar to theirs, considering images and point clouds with an interval of 20 m (instead of 5 m) for the query runs and 10 m for

Table 7.1: Best model vs all runs

	recall@1%			
	2D-to-2D	3D-to-2D	2D-to-3D	3D-to-3D
Ours	96,63	70,44	77,28	98,43

	recall@1			
	2D-to-2D	3D-to-2D	2D-to-3D	3D-to-3D
Ours	88,40	29,51	41,92	93,99

	recall@5			
	2D-to-2D	3D-to-2D	2D-to-3D	3D-to-3D
Ours	95,76	54,79	64,34	97,45

In this table we show the retrieval performances (%) of our approach, computed over all the pairs of runs. We report the recall@1%, recall@1, and recall@5, in all the four possible modalities, *e.g.,* 2D-to-3D represents 2D queries with respect to 3D database.

Table 7.2: Comparison with PointNetVLAD [219]

	recall@1%			
	2D-to-2D	3D-to-2D	2D-to-3D	3D-to-3D
Ours	91.19	55.33	67.27	**94.44**
PNV	-	-	-	80.09

	recall@1			
	2D-to-2D	3D-to-2D	2D-to-3D	3D-to-3D
Ours	81.32	29.92	42.83	**88.98**
PNV	-	-	-	63.33

Retrieval performances comparison between our approach and PointNetVLAD [219]. Both recall@1% and recall@1 are reported.

the database runs, over the same 45 runs, and considering the retrieval as successful if the retrieved place is within 25 m (instead of 20 m).

Table 7.3: Ablation Study

				Modality			
Test	**ASPP**	**Mirroring**	**PC Augm.**	2D-to-2D	3D-to-2D	2D-to-3D	3D-to-3D
Base	✗	✗	✗	96.67	61.53	73.17	96.92
1	✓	✗	✗	96.67	65.31	73.99	97.75
2	✓	✓	✗	96.67	66.30	76.76	97.66
3	✓	✓	✓	96.67	**71.10**	**79.10**	**98.44**

Retrieval performances (%) of our approach in terms of recall@1% by varying different components of the system.

Finally, we performed an ablation study to investigate the effect of different components of the system. In particular, we tested the system without the data augmentation techniques (mirroring and point clouds transformation), and without the ASPP. In these tests, we only considered a subset of 10 runs, to have a quick insight on the effect of the components. The results, reported in Table 7.3, show that both data augmentation, and the ASPP bring substantial improvements to the performance.

Furthermore, once we found the combination which provides the best results, we challenged the robustness of our approach by considering different weather conditions between EVs database and the queries provided to the trained system. For example, we generated a database from 3D samples gathered during summer, and then we provide to the model images acquired during winter. This approach has also been exploited by considering different lighting conditions. In Figure 7.2 we show the recall@k of the four modalities with different values of k. In particular, a query run recorded during summer, one with snow conditions and one recorded during night are compared against a "overcast" database. Some successful and unsuccessful retrieval examples are depicted in Figures 7.3 and 7.4 respectively.

7.3.5 Further Improvement Attempts

Having reached satisfactory results, we tried to increase our model's performance by varying different sub-components of the system. Starting from the architecture type, we tested different backbones such as VGG-16 and ResNet18 for what concerns the 2D data, and PointNet, EdgeConv and SECOND for the 3D data. Furthermore, we investigated various learning methods together with different loss functions and distance measures.

Table 7.4: Recall@1% varying different sub-components

Backbone				Modality			
2D	3D	Loss	Dist.	2D-to-2D	3D-to-2D	2D-to-3D	3D-to-3D
Resnet18	PointNet	(1)	MSE	51.13	20.61	9.93	95.30
VGG-16	PointNet	(1)	MSE	85.51	31.98	51.32	94.53
VGG-16	PointNet	(2)	MSE	86.62	38.77	53.46	95.61
VGG-16	PointNet	(2)	Cosine	88.03	36.85	52.16	96.31
VGG-16	PointNet	(2)	L2	83.38	36.48	47.18	94.64
VGG-16	PointNet	(3)	L2	96.64	31.33	28.69	92.00
VGG-16	EdgeConv	(3)	L2	96.60	67.52	59.50	97.21
VGG-16	SECOND	(2)	MSE	89.27	53.90	59.75	97.10

Loss function are defined as follows: (1) $\mathcal{L}_{trp}^{SM} + \mathcal{L}_{trp}^{CM}$, (2) $\mathcal{L}_{trp}^{SM} + \mathcal{L}_{trp}^{CM} + \mathcal{L}^{JE}$ and (3) is the teacher/student method.

Despite our best efforts, the results in Table 7.4 show that these variations did not improve the performance of the model, instead they reduced the recall capability of our system.

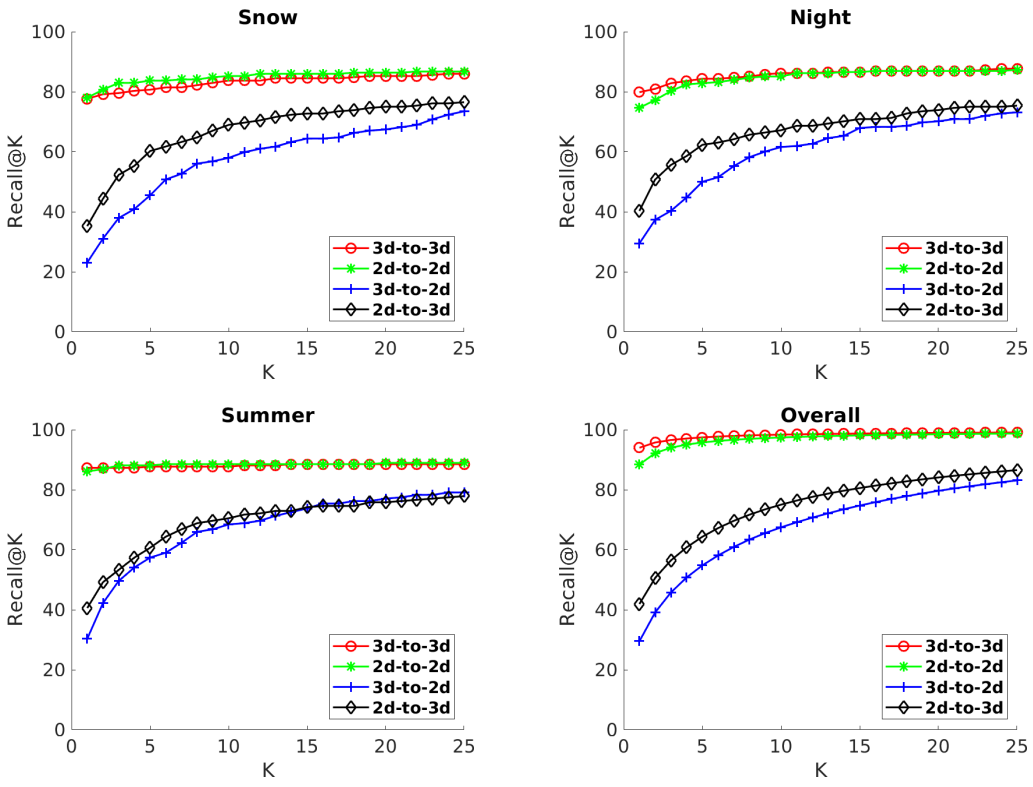


Figure 7.2: Recall measures considering up to $K = 25$ similar places. Here challenging weather and time conditions, *i.e.,* Snow, Night and Summer (direct sunlight), were compared against an "overcast" database. We also provide the plot of the average recall using all available dataset runs.

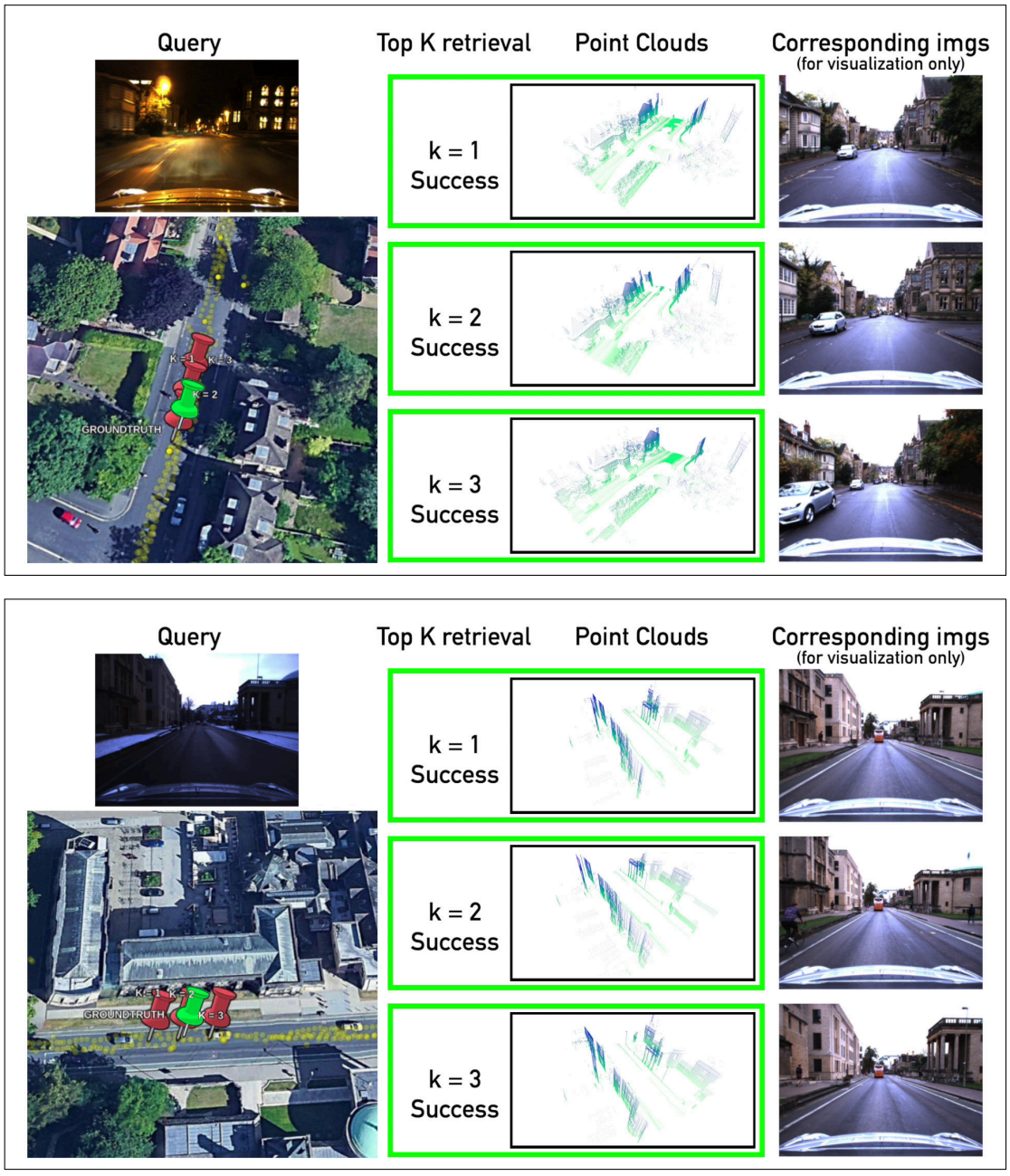


Figure 7.3: Two successful examples of the proposed approach. Each example depicts the query image on top left, the top three retrieved point clouds in the middle with the respective images on the right. Only the point clouds are used for the retrieval, the images on the right are shown only for visualization purposes. The locations of the query and of the retrieved point clouds are depicted on bottom left with green and red pinpoints respectively.

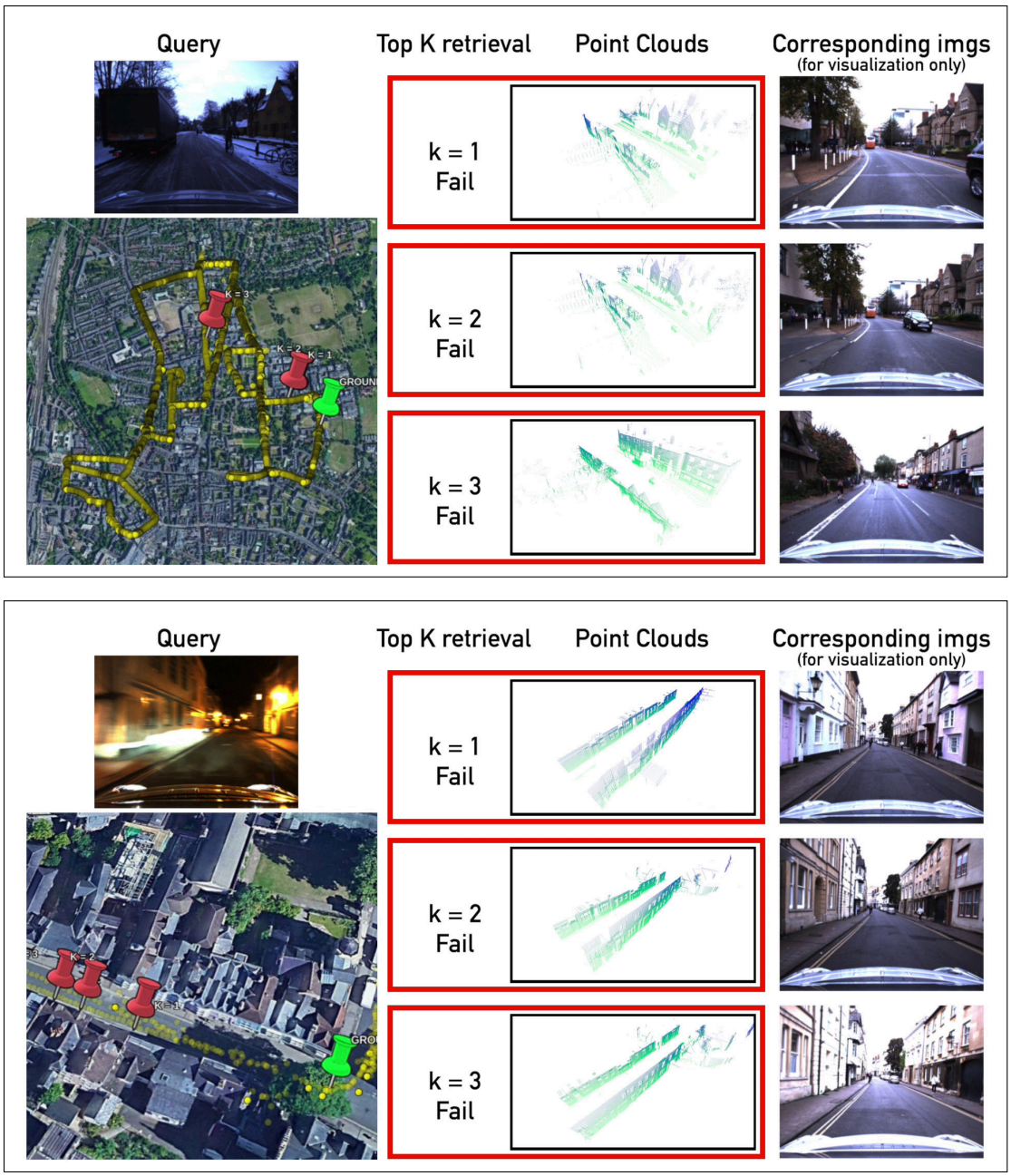


Figure 7.4: Two unsuccessful retrieval. See Figure 7.3 for the description of each example. Even though the approach was unable to retrieve the correct place, in both examples the scene geometry of the retrieved point clouds is very similar to the one in the query.

7.4 Conclusions

In this chapter, we described a novel DNN-based approach to perform global visual localization in LiDAR-maps. In particular, we proposed to jointly train a 2D CNN and a 3D DNN to produce a shared embedding space. We trained and validated our DNNs on the challenging Oxford RobotCar dataset, including all weather conditions, and promising results show the effectiveness of our approach. We also obtained comparable performance on 2D-to-2D and state-of-the-art results on 3D-to-3D modalities, although this was not the focus of our research. To our knowledge, an approach that performs a global visual localization using LiDAR-maps has never been presented before.

Chapter 8

Conclusions

This thesis presented different approaches for autonomous vehicles' localization in highway and urban environments, using only cheap camera sensors onboard. In the first part of this thesis, we integrated CNNs and machine learning techniques in different pipelines for localizing the vehicle with respect to the OSM service by matching high-level features (road lanes, road intersections, and buildings). In the second part, we proposed two end-to-end DNN-based approaches for local and global localization, respectively, that matches a single RGB image with a LiDAR-map of the working environments. The decision to switch from OSM to LiDAR-maps is that we believe that in the near future, map-making companies (such as TomTom and HERE) will release the so-called HD-maps. These maps are specifically designed for autonomous driving applications, and they will provide a geometric reconstruction similar to the one obtained with LiDAR-maps.

The main contributions of this thesis are listed below.

- We proposed a HMM to estimate the ego-lane of the vehicle in highway-like scenarios exploiting existing line detectors and trackers. Our model improved the localization robustness in conditions where lane markings are missing, hidden by traffic clutter, or difficult to detect because of lighting issues.
- We integrated state-of-the-art CNNs for semantic segmentation and geometric reconstruction in order to detect road geometry and buildings. We matched these high-level features with their counterparts in OSM to localize the vehicle in a probabilistic fashion by means of a particle filtering technique.
- We introduced CMRNet, a novel end-to-end DNN for vehicle localization in LiDAR-maps that achieved lane-level localization accuracy in challenging conditions (such as rain, snow, and night). Differently from state-of-the-art DNN-based approaches

for camera localization, CMRNet does not learn the map, but instead it learns to match images to the map. Therefore, it can be used in any environment for which a LiDAR-map is available. Moreover, CMRNet achieved real-time capabilities, make it suitable for autonomous driving applications.

- Finally, we proposed a novel DNN-based technique for global visual localization in LiDAR-maps. We jointly trained two DNNs, one for images and the other for point clouds, to create a shared embedding space. This embedding space allows us to perform place recognition with heterogeneous sensors.

Bibliography

[1] W. H. Organization. *Global status report on road safety 2018*. 2018. URL: https://www.who.int/violence_injury_prevention/road_safety_status/2018/en/.

[2] Y. Gu, L. T. Hsu, and S. Kamijo. "GNSS/Onboard Inertial Sensor Integration With the Aid of 3-D Building Map for Lane-Level Vehicle Self-Localization in Urban Canyon". In: *IEEE Transactions on Vehicular Technology* 65.6 (June 2016), pp. 4274–4287.

[3] K. He et al. "Deep residual learning for image recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[4] O. Ronneberger, P. Fischer, and T. Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[5] Y. Kim. "Convolutional Neural Networks for Sentence Classification". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1746–1751.

[6] D. Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), p. 484.

[7] S. Ren et al. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[8] J. Redmon and A. Farhadi. "YOLO9000: better, faster, stronger". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 7263–7271.

[9] W. Liu et al. "Ssd: Single shot multibox detector". In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[10] J. Long, E. Shelhamer, and T. Darrell. "Fully convolutional networks for semantic segmentation". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3431–3440.

[11] L.-C. Chen et al. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2018).

[12] H. Zhao et al. "Pyramid scene parsing network". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[13] X. Cheng, P. Wang, and R. Yang. "Learning depth with convolutional spatial propagation network". In: *arXiv preprint arXiv:1810.02695* (2018).

[14] Z. Yin, T. Darrell, and F. Yu. "Hierarchical Discrete Distribution Decomposition for Match Density Estimation". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 6044–6053.

[15] J.-R. Chang and Y.-S. Chen. "Pyramid stereo matching network". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 5410–5418.

[16] J. Pang et al. "Cascade residual learning: A two-stage convolutional neural network for stereo matching". In: *Workshop on Geometry Meets Deep Learning (ICCVW 2017)*. 2017.

[17] D. Sun et al. "PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[18] A. Dosovitskiy et al. "Flownet: Learning optical flow with convolutional networks". In: *IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 2758–2766.

[19] E. Ilg et al. "Flownet 2.0: Evolution of optical flow estimation with deep networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2462–2470.

[20] W.-C. Ma et al. "Deep Rigid Instance Scene Flow". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 3614–3622.

[21] A. Kendall, M. Grimes, and R. Cipolla. "PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization". In: *IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015.

[22] S. Brahmbhatt et al. "Geometry-Aware Learning of Maps for Camera Localization". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[23] N. Radwan, A. Valada, and W. Burgard. "VLocNet++: Deep Multitask Learning for Semantic Visual Localization and Odometry". In: *IEEERobotics and Automation Letters* 3.4 (2018), pp. 4407–4414.

[24] J. Levinson and S. Thrun. "Robust vehicle localization in urban environments using probabilistic maps". In: *IEEE International Conference on Robotics and Automation (ICRA)*. May 2010, pp. 4372–4378.

[25] E. D. Dickmanns and B. D. Mysliwetz. "Recursive 3-D Road and Relative Ego-State Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), pp. 199–213.

[26] D. Pomerleau. "RALPH: rapidly adapting lateral position handler". In: *Proc. Intell. Veh. '95 Symp.* 1995, pp. 506–511.

[27] A. Broggi et al. "The ARGO autonomous vehicle's vision and control systems". In: *International Journal of Intelligent Control and Systems* 3.4 (1999), pp. 409–441.

[28] S. Thrun et al. "Stanley: The robot that won the DARPA Grand Challenge". In: *Journal of field Robotics* 23.9 (2006), pp. 661–692.

[29] C. Urmson et al. "Autonomous driving in urban environments: Boss and the urban challenge". In: *Journal of Field Robotics* 25.8 (2008), pp. 425–466.

[30] M. Nájar et al. "Mobile location with bias tracking in non-line-of-sight". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Vol. 3. IEEE. 2004, pp. iii–956.

[31] J. M. Huerta et al. "Joint particle filter and UKF position tracking in severe non-line-of-sight situations". In: *IEEE Journal of Selected Topics in Signal Processing* 3.5 (2009), pp. 874–888.

[32] Y. Ng and G. X. Gao. “Direct position estimation utilizing non-line-of-sight (NLOS) GPS signals”. In: *ION GNSS*. 2016, pp. 1279–1284.

[33] M. Adjrad and P. D. Groves. “Intelligent urban positioning using shadow matching and GNSS ranging aided by 3D mapping”. In: Institute of Navigation (ION). 2016.

[34] D. M. Cole and P. M. Newman. “Using laser range data for 3D SLAM in outdoor environments”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2006, pp. 1556–1563.

[35] A. Nüchter et al. “6D SLAM—3D mapping outdoor environments”. In: *Journal of Field Robotics* 24.8-9 (2007), pp. 699–722.

[36] R. Mur-Artal and J. D. Tardós. “Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras”. In: *IEEE Transactions on Robotics* 33.5 (2017), pp. 1255–1262.

[37] C. Kerl, J. Sturm, and D. Cremers. “Dense visual SLAM for RGB-D cameras”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2013, pp. 2100–2106.

[38] J. Engel, T. Schöps, and D. Cremers. “LSD-SLAM: Large-scale direct monocular SLAM”. In: *European conference on computer vision*. Springer. 2014, pp. 834–849.

[39] M. Pauly, M. Gross, and L. P. Kobbelt. “Efficient simplification of point-sampled surfaces”. In: *Proceedings of the conference on Visualization'02*. IEEE Computer Society. 2002, pp. 163–170.

[40] C. Moenning and N. A. Dodgson. “A new point cloud simplification algorithm”. In: *Proc. Int. Conf. on Visualization, Imaging and Image Processing*. 2003, pp. 1027–1033.

[41] C. Tomasi and T. Kanade. “Shape and motion from image streams under orthography: a factorization method”. In: *International journal of computer vision* 9.2 (1992), pp. 137–154.

[42] C. Wu. “Towards linear-time incremental structure from motion”. In: *2013 International Conference on 3D Vision-3DV 2013*. IEEE. 2013, pp. 127–134.

[43] D. G. Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *International Journal of Computer Vision* 60.2 (Nov. 2004), pp. 91–110.

[44] H. Bay et al. “Speeded-Up Robust Features (SURF)”. In: *Computer Vision and Image Understanding* 110.3 (2008), pp. 346–359.

[45] B. Triggs et al. “Bundle adjustment—a modern synthesis”. In: *International workshop on vision algorithms*. Springer. 1999, pp. 298–372.

[46] S. Thrun, W. Burgard, and D. Fox. “A probabilistic approach to concurrent mapping and localization for mobile robots”. In: *Autonomous Robots* 5.3-4 (1998), pp. 253–271.

[47] F. Dellaert et al. “Monte carlo localization for mobile robots”. In: *ICRA*. Vol. 2. 1999, pp. 1322–1328.

[48] J. Bares et al. “Ambler: An autonomous rover for planetary exploration”. In: *Computer* 22.6 (1989), pp. 18–26.

[49] C. F. Olson. “Probabilistic self-localization for mobile robots”. In: *IEEE Transactions on Robotics and Automation* 16.1 (2000), pp. 55–66.

[50] P. Pfaff, R. Triebel, and W. Burgard. “An efficient extension to elevation maps for outdoor terrain mapping and loop closing”. In: *The International Journal of Robotics Research* 26.2 (2007), pp. 217–230.

[51] R. Triebel, P. Pfaff, and W. Burgard. “Multi-level surface maps for outdoor terrain mapping and loop closing”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2006, pp. 2276–2282.

[52] A. Hornung et al. “OctoMap: An efficient probabilistic 3D mapping framework based on octrees”. In: *Autonomous robots* 34.3 (2013), pp. 189–206.

[53] D. Meagher. “Geometric modeling using octree encoding”. In: *Computer graphics and image processing* 19.2 (1982), pp. 129–147.

[54] M. Haklay and P. Weber. “Openstreetmap: User-generated street maps”. In: *IEEE Pervasive Computing* 7.4 (2008), pp. 12–18.

[55] G. Mattyus, W. Luo, and R. Urtasun. “DeepRoadMapper: Extracting Road Topology From Aerial Images”. In: *IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[56] D. Costea et al. “Creating roadmaps in aerial images with generative adversarial networks and smoothing-based optimization”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2100–2109.

[57] G. Mattyus et al. “Enhancing Road Maps by Parsing Aerial Images Around the World”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2015.

[58] G. Mattyus et al. “HD Maps: Fine-grained Road Segmentation by Parsing Ground and Aerial Images”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[59] M. Hentschel and B. Wagner. “Autonomous robot navigation based on OpenStreetMap geodata”. In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. Sept. 2010, pp. 1645–1650.

[60] O. Vysotska and C. Stachniss. “Improving SLAM by exploiting building information from publicly available maps and localization priors”. In: *PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Science* 85.1 (2017), pp. 53–65.

[61] M. A. Brubaker, A. Geiger, and R. Urtasun. “Lost! leveraging the crowd for probabilistic visual self-localization”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013, pp. 3057–3064.

[62] G. Floros, B. van der Zander, and B. Leibe. “OpenStreetSLAM: Global vehicle localization using OpenStreetMaps”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. May 2013, pp. 1054–1059.

[63] P. Ruchti et al. “Localization on OpenStreetMap data using a 3D laser scanner”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. May 2015, pp. 5260–5265.

[64] W. S. McCulloch and W. Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[65] B. Widrow and M. E. Hoff. *Adaptive switching circuits*. Tech. rep. Stanford Univ Ca Stanford Electronics Labs, 1960.

[66] F. Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[67] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969.

[68] S. Linnainmaa. "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors". In: *Master's Thesis (in Finnish), Univ. Helsinki* (1970), pp. 6–7.

[69] K. Hornik, M. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[70] Y. LeCun et al. "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4 (1989), pp. 541–551.

[71] D. A. Pomerleau. "Alvinn: An autonomous land vehicle in a neural network". In: *Advances in neural information processing systems*. 1989, pp. 305–313.

[72] L.-J. Lin. *Reinforcement learning for robots using neural networks*. Tech. rep. Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[73] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[74] B. E. Boser, I. M. Guyon, and V. N. Vapnik. "A training algorithm for optimal margin classifiers". In: *Proceedings of the fifth annual workshop on Computational learning theory*. ACM. 1992, pp. 144–152.

[75] Y. LeCun et al. "Comparison of learning algorithms for handwritten digit recognition". In: *International conference on artificial neural networks*. Vol. 60. Perth, Australia. 1995, pp. 53–60.

[76] G. E. Hinton, S. Osindero, and Y.-W. Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[77] K. Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological cybernetics* 36.4 (1980), pp. 193–202.

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

[79] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[80] O. Russakovsky et al. “Imagenet large scale visual recognition challenge”. In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[81] A. L. Maas, A. Y. Hannun, and A. Y. Ng. “Rectifier nonlinearities improve neural network acoustic models”. In: *Proc. icml*. Vol. 30. 1. 2013, p. 3.

[82] V. Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), p. 529.

[83] J. J. Hopfield. “Neural networks and physical systems with emergent collective computational abilities”. In: *Proceedings of the national academy of sciences* 79.8 (1982), pp. 2554–2558.

[84] A. Graves, A.-r. Mohamed, and G. Hinton. “Speech recognition with deep recurrent neural networks”. In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 6645–6649.

[85] I. Sutskever, O. Vinyals, and Q. Le. “Sequence to sequence learning with neural networks”. In: *Advances in NIPS* (2014).

[86] K. Xu et al. “Show, attend and tell: Neural image caption generation with visual attention”. In: *International conference on machine learning*. 2015, pp. 2048–2057.

[87] K. Gregor et al. “Draw: A recurrent neural network for image generation”. In: *arXiv preprint arXiv:1502.04623* (2015).

[88] K. He et al. “Mask r-cnn”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2961–2969.

[89] S. Wang et al. “Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 2043–2050.

[90] N. Yang et al. “Deep Virtual Stereo Odometry: Leveraging Deep Depth Prediction for Monocular Direct Sparse Odometry”. In: *Computer Vision – ECCV 2018*. Ed. by V. Ferrari et al. Cham: Springer International Publishing, 2018, pp. 835–852.

[91] H. Zhan et al. “Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 340–349.

[92] A. Kendall and R. Cipolla. "Geometric Loss Functions for Camera Pose Regression With Deep Learning". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[93] A. Kendall and R. Cipolla. "Modelling uncertainty in deep learning for camera relocalization". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 4762–4769.

[94] T. Naseer and W. Burgard. "Deep regression for monocular camera-based 6-DoF global localization in outdoor environments". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 1525–1530.

[95] R. Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 580–587.

[96] R. Girshick. "Fast r-cnn". In: *IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448.

[97] J. Redmon et al. "You only look once: Unified, real-time object detection". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788.

[98] L.-C. Chen et al. "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation". In: *Computer Vision – ECCV 2018*. Ed. by V. Ferrari et al. Cham: Springer International Publishing, 2018, pp. 833–851.

[99] Y. Zhu et al. "Improving Semantic Segmentation via Video Propagation and Label Relaxation". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 8856–8865.

[100] S.-L. Yu et al. "Vehicle detection and localization on bird's eye view elevation images using convolutional neural network". In: *IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE. 2017, pp. 102–109.

[101] B. Yang, W. Luo, and R. Urtasun. "Pixor: Real-time 3d object detection from point clouds". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 7652–7660.

[102] S. Shi, X. Wang, and H. Li. "Pointrcnn: 3d object proposal generation and detection from point cloud". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 770–779.

[103] Y. Zhou and O. Tuzel. "Voxelnet: End-to-end learning for point cloud based 3d object detection". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 4490–4499.

[104] Y. Yan, Y. Mao, and B. Li. "Second: Sparsely embedded convolutional detection". In: *Sensors* 18.10 (2018), p. 3337.

[105] X. Chen et al. "Multi-View 3D Object Detection Network for Autonomous Driving". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[106] C. R. Qi et al. "Frustum pointnets for 3d object detection from rgb-d data". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 918–927.

[107] J. Ku et al. "Joint 3d proposal generation and object detection from view aggregation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1–8.

[108] Z. Wu et al. "3d shapenets: A deep representation for volumetric shapes". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1912–1920.

[109] D. Maturana and S. Scherer. "Voxnet: A 3d convolutional neural network for real-time object recognition". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 922–928.

[110] C. R. Qi et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[111] C. R. Qi et al. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space". In: *Advances in neural information processing systems*. 2017, pp. 5099–5108.

[112] Y. Wang et al. "Dynamic Graph CNN for Learning on Point Clouds". In: *ACM Transactions on Graphics (TOG)* 38.5 (2019), pp. 1–12.

[113] R. Klokov and V. Lempitsky. "Escape from cells: Deep kd-networks for the recognition of 3d point cloud models". In: *IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 863–872.

[114] P.-S. Wang et al. "O-cnn: Octree-based convolutional neural networks for 3d shape analysis". In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), p. 72.

[115] A. Geiger et al. "Vision meets Robotics: The KITTI Dataset". In: *International Journal of Robotics Research (IJRR)* (2013).

[116] H. Fu et al. "Deep ordinal regression network for monocular depth estimation". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 2002–2011.

[117] C. Godard, O. Mac Aodha, and G. J. Brostow. "Unsupervised monocular depth estimation with left-right consistency". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 270–279.

[118] Y. Kuznietsov, J. Stuckler, and B. Leibe. "Semi-supervised deep learning for monocular depth map prediction". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 6647–6655.

[119] A. Bar Hillel et al. "Recent progress in road and lane detection: a survey". In: *Mach. Vis. Appl.* 25.3 (Apr. 2014), pp. 727–745.

[120] M. Bertozzi and A. Broggi. "GOLD: a parallel real-time stereo vision system for generic obstacle and lane detection". In: *IEEE Transactions on Image Processing* 7.1 (1998), pp. 62–81.

[121] C. J. Taylor, J. Malik, and J. Weber. "A real-time approach to stereopsis and lane-finding". In: *IEEE Intelligent Vehicles Symposium (IV)*. 1996, pp. 207–212.

[122] Y. Wang, E. K. Teoh, and D. Shen. "Lane detection and tracking using B-Snake". In: *Image Vis. Comput.* 22.4 (2004), pp. 269–280.

[123] J. McCall and M. Trivedi. "Video-Based Lane Estimation and Tracking for Driver Assistance: Survey, System, and Evaluation". In: *IEEE Transactions on Intelligent Transportation Systems* 7.1 (Mar. 2006), pp. 20–37.

[124] F. Kuhnt et al. "Lane-precise Localization of Intelligent Vehicles Using the Surrounding Object Constellation". In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)* November (2016).

[125] G. Cao et al. “Camera to map alignment for accurate low-cost lane-level scene interpretation”. In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. Nov. 2016.

[126] T. Gao and H. Aghajan. “Self lane assignment using egocentric smart mobile camera for intelligent GPS navigation”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009, pp. 57–62.

[127] Z. Kim. “Robust lane detection and tracking in challenging scenarios”. In: *IEEE Transactions on Intelligent Vehicles* 9.1 (2008), pp. 16–26.

[128] T. Kuhnl, F. Kummert, and J. Fritsch. “Visual ego-vehicle lane assignment using Spatial Ray features”. In: *IEEE Intelligent Vehicles Symposium (IV)*. Iv. 2013, pp. 1101–1106.

[129] J. Rabe, M. Necker, and C. Stiller. “Ego-lane estimation for lane-level navigation in urban scenarios”. In: *IEEE Intelligent Vehicles Symposium (IV)*. Vol. 2016-Augus. Iv. IEEE, June 2016, pp. 896–901.

[130] S. Lee, S. W. Kim, and S. W. Seo. “Accurate ego-lane recognition utilizing multiple road characteristics in a Bayesian network framework”. In: *IEEE Intelligent Vehicles Symposium (IV)*. Vol. 2015-Augus. Iv. 2015, pp. 543–548.

[131] M. Nieto et al. “Robust multiple lane road modeling based on perspective analysis”. In: *Int. Conf. Image Process. ICIP*. 2008, pp. 2396–2399.

[132] Y. Jiang, F. Gao, and G. Xu. “Computer vision-based multiple-lane detection on straight road and in a curve”. In: *Int. Conf. Image Anal. Signal Process.* Vol. 2. 2010, pp. 114–117.

[133] S.-N. Kang et al. “Multi-lane detection based on accurate geometric lane estimation in highway scenarios”. In: *IEEE Intelligent Vehicles Symposium (IV)*. Iv. IEEE, June 2014, pp. 221–226.

[134] J. Kim and M. Lee. “Robust lane detection based on convolutional neural network and random sample consensus”. In: *International conference on neural information processing*. Springer. 2014, pp. 454–461.

[135] D. Neven et al. “Towards end-to-end lane detection: an instance segmentation approach”. In: *IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2018, pp. 286–291.

[136] S. Lee et al. “Vpgnet: Vanishing point guided network for lane and road marking detection and recognition”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 1947–1955.

[137] L.-C. Chen et al. “Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation”. In: *Computer Vision – ECCV 2018*. Ed. by V. Ferrari et al. Cham: Springer International Publishing, 2018, pp. 833–851.

[138] X. Pan et al. “Spatial as deep: Spatial cnn for traffic scene understanding”. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[139] Y. Hou et al. “Learning lightweight lane detection cnns by self attention distillation”. In: *arXiv preprint arXiv:1908.00821* (2019).

[140] S. Suzuki. “Topological structural analysis of digitized binary images by border following”. In: *Computer Vision, Graphics, and Image Processing* 30.1 (1985), pp. 32–46.

[141] H. Hirschmüller. “Accurate and efficient stereo processing by semi-global matching and mutual information”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2005.

[142] A. Geiger, M. Roser, and R. Urtasun. “Efficient Large-Scale Stereo Matching”. In: *Asian Conference on Computer Vision (ACCV)*. 2010.

[143] M. Aly. “Real time detection of lane markers in urban streets”. In: *IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2008, pp. 7–12.

[144] J. Hur. “Multi-Lane Detection in Highway and Urban Driving Environment”. MA thesis. Seoul National University, Dept. of Electrical and Computer Engineering, 2013.

[145] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach, sect. 15.5*. 2nd ed. Pearson Education, 2003.

[146] J. Bilmes. *On virtual evidence and soft evidence in Bayesian networks*. Tech. rep. retr. 09.Jan.18 at `https://www2.ee.washington.edu/techsite/papers/refer/UWEETR-2004-0016.html`. Washington Univ., 2004.

[147] D. Obradovic, H. Lenz, and M. Schupfner. “Fusion of Sensor Data in Siemens Car Navigation System”. In: *IEEE Transactions on Vehicular Technology* (Jan. 2007).

[148] I. P. Alonso et al. “Accurate Global Localization Using Visual Odometry and Digital Maps on Urban Environments”. In: *IEEE Transactions on Intelligent Transportation Systems* (Dec. 2012).

[149] M. Raaijmakers and M. E. Bouzouraa. “In-vehicle Roundabout Perception Supported by A Priori Map Data”. In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. Sept. 2015.

[150] S. Nedevschi et al. “Accurate Ego-Vehicle Global Localization at Intersections Through Alignment of Visual Data With Digital Map”. In: *IEEE Transactions on Intelligent Transportation Systems* (June 2013).

[151] A. Vu et al. “Real-Time Computer Vision/DGPS-Aided Inertial Navigation System for Lane-Level Vehicle Navigation”. In: *IEEE Transactions on Vehicular Technology* (June 2012).

[152] G. Floros and B. Leibe. “Joint 2D-3D temporally consistent semantic segmentation of street scenes”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2012.

[153] C. Fernández et al. “A Comparative Analysis of Decision Trees Based Classifiers for Road Detection in Urban Environments”. In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. Sept. 2015.

[154] B. Hummel, W. Thiemann, and I. Lulcheva. “Scene Understanding of Urban Road Intersections with Description Logic”. In: *Logic and Probability for Scene Interpretation*. 2008.

[155] A. Gupta, A. A. Efros, and M. Hebert. “Blocks World Revisited: Image Understanding Using Qualitative Geometry and Mechanics”. In: *Computer Vision – ECCV 2010*. Ed. by K. Daniilidis, P. Maragos, and N. Paragios. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 482–496.

[156] D. Hoiem, A. A. Efros, and M. Hebert. “Closing the loop in scene interpretation”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2008.

[157] J. M. Álvarez et al. “Combining Priors, Appearance, and Context for Road Detection”. In: *IEEE Transactions on Intelligent Transportation Systems* (June 2014).

[158] E.-H. Choi. *Crash factors in intersection-related crashes: An on-scene perspective*. Tech. rep. 2010.

[159] A. Geiger, M. Lauer, and R. Urtasun. "A generative model for 3D urban scene understanding from movable platforms". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2011.

[160] A. Barth, J. Siegemund, and J. Schwehr. "Fast and precise localization at stop intersections". In: *IEEE Intelligent Vehicles Symposium (IV)*. June 2013.

[161] T. R. Kushner and S. Puri. "Progress In Road Intersection Detection For Autonomous Vehicle Navigation". In: *Proc. SPIE*. 1987.

[162] J. D. Crisman and C. E. Thorpe. "SCARF: a color vision system that tracks roads and intersections". In: *IEEE Transactions on Robotics and Automation* (Feb. 1993).

[163] T. M. Jochem, D. A. Pomerleau, and C. E. Thorpe. "Vision-based neural network road and intersection detection and traversal". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Aug. 1995.

[164] Q. Zhu et al. "3d lidar point cloud based intersection recognition for autonomous driving". In: *IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2012, pp. 456–461.

[165] A. Y. Hata et al. "Road geometry classification using ANN". In: *IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2014, pp. 1319–1324.

[166] L. Wang et al. "3D-LIDAR based branch estimation and intersection location for autonomous vehicles". In: *IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 1440–1445.

[167] A. L. Ballardini et al. "A Framework for Outdoor Urban Environment Estimation". In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. Sept. 2015.

[168] F. Yu and V. Koltun. "Multi-Scale Context Aggregation by Dilated Convolutions". In: *ICLR*. 2016.

[169] M. Cordts et al. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[170] Z. Deng. *PyTorch for Semantic Segmentation*. 2017. URL: %7Bhttps://github.com/zijundeng/pytorch-semantic-segmentation%7D.

[171] H. Alhaija et al. "Augmented Reality Meets Computer Vision: Efficient Data Generation for Urban Driving Scenes". In: 2018.

[172] F. Smarandache and J. Dezert. *Advances and Applications of DSmT for Information Fusion, vol. III*. American Research Press, 2009.

[173] A. Geiger, J. Ziegler, and C. Stiller. "StereoScan: Dense 3D Reconstruction in Real-time". In: *IEEE Transactions on Intelligent Vehicles*. 2011.

[174] S. Kong and C. Fowlkes. "Pixel-wise attentional gating for parsimonious pixel labeling". In: *arXiv preprint arXiv:1805.01556* (2018).

[175] M. Ochs, A. Kretz, and R. Mester. "SDNet: Semantically Guided Depth Estimation Network". In: *arXiv preprint arXiv:1907.10659* (2019).

[176] G. Yang et al. "SegStereo: Exploiting Semantic Information for Disparity Estimation". In: *The European Conference on Computer Vision (ECCV)*. Sept. 2018.

[177] I. Kreso, S. Segvic, and J. Krapac. "Ladder-style densenets for semantic segmentation of large natural images". In: *IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 238–245.

[178] Y. Zhu et al. "Improving Semantic Segmentation via Video Propagation and Label Relaxation". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[179] A. Geiger, P. Lenz, and R. Urtasun. "Are we ready for autonomous driving? The KITTI vision benchmark suite". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2012, pp. 3354–3361.

[180] J.-L. Blanco-Claraco, F.-Á. Moreno-Dueñas, and J. González-Jiménez. "The Málaga urban dataset: High-rate stereo and LiDAR in a realistic urban scenario". In: *The International Journal of Robotics Research* 33.2 (2014), pp. 207–214.

[181] W. Maddern et al. "1 Year, 1000km: The Oxford RobotCar Dataset". In: *The International Journal of Robotics Research (IJRR)* (2017).

[182] G. Fontana, M. Matteucci, and D. G. Sorrenti. "Rawseeds: building a benchmarking toolkit for autonomous robotics". In: *Methods and Experimental Techniques in Computer Engineering*. Springer, 2014, pp. 55–68.

[183] X. Huang et al. “The apolloscape dataset for autonomous driving”. In: *IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 954–960.

[184] Z. Tao et al. “Mapping and localization using GPS, lane markings and proprioceptive sensors”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Nov. 2013, pp. 406–412.

[185] M. Schreiber, C. Knöppel, and U. Franke. “LaneLoc: Lane marking based localization using highly accurate maps”. In: *IEEE Intelligent Vehicles Symposium (IV)*. June 2013, pp. 449–454.

[186] R. Spangenberg, D. Goehring, and R. Rojas. “Pole-based localization for autonomous vehicles in urban scenarios”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 2161–2166.

[187] L. Wei, B. Soheilian, and V. Gouet-Brunet. “Augmenting vehicle localization accuracy with cameras and 3d road infrastructure database”. In: *European Conference on Computer Vision*. Springer. 2014, pp. 194–208.

[188] J. J. Corso. “Discriminative modeling by Boosting on Multilevel Aggregates”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2008, pp. 1–8.

[189] D. Larnaout et al. “Vision-Based Differential GPS: Improving VSLAM / GPS Fusion in Urban Environment with 3D Building Models”. In: *IEEE International Conference on 3D Vision (3DV)*. Vol. 1. Dec. 2014, pp. 432–439.

[190] P. David. *Detecting Planar Surfaces in Outdoor Urban Environments*. Tech. rep. ARL-TR-4599. United States Army Research Laboratory DTIC Document, Sept. 2008.

[191] J. A. Delmerico, P. David, and J. J. Corso. “Building facade detection, segmentation, and parameter estimation for mobile robot localization and guidance”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2011, pp. 1632–1639.

[192] R. Gadde et al. “Efficient 2D and 3D Facade Segmentation Using Auto-Context”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.5 (2018), pp. 1273–1280.

[193] A. Fond, M.-O. Berger, and G. Simon. "Facade Proposals for Urban Augmented Reality". In: *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 2017, pp. 32–41.

[194] R. Fathalla and G. Vogiatzis. "A Deep Learning Pipeline for Semantic Facade Segmentation". en. In: *British Machine Vision Conference (BMVC*. `http://www.bmva.org/bmvc/2017/papers/paper120/index.html`. London, UK: British Machine Vision Association, 2017, p. 120.

[195] H. Su et al. "SPLATNet: Sparse Lattice Networks for Point Cloud Processing". en. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. `https://ieeexplore.ieee.org/document/8578366/`. Salt Lake City, UT: IEEE, 2018, pp. 2530–2539.

[196] N. Haala and J. Böhm. "A multi-sensor system for positioning in urban environments". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 58.1-2 (2003), pp. 31–42.

[197] S. Zhu et al. "Video/GIS registration system based on skyline matching method". In: *IEEE International Conference on Image Processing*. IEEE. 2013, pp. 3632–3636.

[198] M. Bansal, K. Daniilidis, and H. Sawhney. "Ultrawide baseline facade matching for geo-localization". In: *Large-Scale Visual Geo-Localization*. Springer, 2016, pp. 77–98.

[199] P. J. Besl and N. D. McKay. "A method for registration of 3-D shapes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (Feb. 1992), pp. 239–256.

[200] Y. Chen and G. Medioni. "Object modelling by registration of multiple range images". In: *Image and Vis. Comp.* 10.3 (1992). Range Image Understanding, pp. 145–155.

[201] A. Segal, D. Haehnel, and S. Thrun. "Generalized-icp". In: *Robotics: science and systems (RSS)*. Vol. 2. 4. 2009, p. 435.

[202] F. Pomerleau et al. "Comparing ICP variants on real-world data sets". In: *Autonomous Robots* 34.3 (Apr. 2013), pp. 133–148.

[203] Q.-Y. Zhou, J. Park, and V. Koltun. "Fast global registration". In: *European Conference on Computer Vision*. Springer. 2016, pp. 766–782.

[204] I. Parra et al. "Visual Odometry and Map Fusion for GPS Navigation Assistance". en. In: *IEEE International Symposium on Industrial Electronics*. `http://ieeexplore.ieee.org/document/5984266/`. Gdansk, Poland: IEEE, 2011, pp. 832–837.

[205] C. Mazzotti, N. Sancisi, and V. Parenti-Castelli. "A Measure of the Distance Between Two Rigid-Body Poses Based on the Use of Platonic Solids". In: *ROMANSY 21 - Robot Design, Dynamics and Control*. Ed. by V. Parenti-Castelli and W. Schiehlen. Cham: Springer International Publishing, 2016, pp. 81–89.

[206] *The future of maps: technologies, processes and ecosystem*. `https://www.here.com/file/7766/download?token=dwOqPUix`. Accessed: 2019-04-13. Dec. 2018.

[207] A. R. Zamir and M. Shah. "Accurate Image Localization Based on Google Maps Street View". In: *Computer Vision – ECCV 2010*. Ed. by K. Daniilidis, P. Maragos, and N. Paragios. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 255–268.

[208] T. Sattler, B. Leibe, and L. Kobbelt. "Improving Image-Based Localization by Active Correspondence Search". In: *Computer Vision – ECCV 2012*. Ed. by A. Fitzgibbon et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 752–765.

[209] J. Shotton et al. "Scene Coordinate Regression Forests for Camera Relocalization in RGB-D Images". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2013.

[210] R. Clark et al. "VidLoc: A Deep Spatio-Temporal Model for 6-DoF Video-Clip Relocalization". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[211] E. Brachmann et al. "DSAC - Differentiable RANSAC for Camera Localization". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[212] E. Brachmann and C. Rother. "Learning Less Is More - 6D Camera Localization via 3D Surface Regression". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[213] T. Caselitz et al. “Monocular camera localization in 3D LiDAR maps”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 1926–1931.

[214] R. W. Wolcott and R. M. Eustice. “Visual localization within LIDAR maps for automated urban driving”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2014, pp. 176–183.

[215] P. Neubert, S. Schubert, and P. Protzel. “Sampling-based methods for visual navigation in 3D maps by synthesizing depth images”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 2492–2498.

[216] N. Schneider et al. “RegNet: Multimodal sensor registration using deep neural networks”. In: *IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 1803–1810.

[217] R. Pintus, E. Gobbetti, and M. Agus. “Real-time rendering of massive unstructured raw point clouds using screen-space operators”. In: *Proceedings of the 12th International conference on Virtual Reality, Archaeology and Cultural Heritage*. Eurographics Association. 2011, pp. 105–112.

[218] A. Paszke et al. “Automatic Differentiation in PyTorch”. In: *NIPS Autodiff Workshop*. 2017.

[219] M. Angelina Uy and G. Hee Lee. “PointNetVLAD: Deep Point Cloud Based Retrieval for Large-Scale Place Recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[220] D. Gálvez-López and J. D. Tardos. “Bags of binary words for fast place recognition in image sequences”. In: *IEEE Transactions on Robotics* 28.5 (2012), pp. 1188–1197.

[221] J. Philbin et al. “Object retrieval with large vocabularies and fast spatial matching”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2007.

[222] O. Chum et al. “Total recall II: Query expansion revisited”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2011.

[223] R. Arandjelovic et al. “NetVLAD: CNN Architecture for Weakly Supervised Place Recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[224] W.-C. Ma et al. “Find your way by observing the sun and other semantic cues”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 6292–6299.

[225] S. Choi, Q.-Y. Zhou, and V. Koltun. “Robust reconstruction of indoor scenes”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 5556–5565.

[226] N. Mellado, D. Aiger, and N. J. Mitra. “Super 4pcs fast global pointcloud registration via smart indexing”. In: *Computer Graphics Forum*. Vol. 33. 5. Wiley Online Library. 2014, pp. 205–215.

[227] M. Bosse and R. Zlot. “Place recognition using keypoint voting in large 3D lidar datasets”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2013, pp. 2677–2684.

[228] R. Dubé et al. “Segmatch: Segment based place recognition in 3d point clouds”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 5266–5272.

[229] B. Williams et al. “A comparison of loop closing techniques in monocular SLAM”. In: *Robotics and Autonomous Systems* 57.12 (2009), pp. 1188–1197.

[230] P. Newman and K. Ho. “SLAM-loop closing with visually salient features”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2005, pp. 635–642.

[231] H. Jégou et al. “Aggregating local descriptors into a compact image representation”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. San Francisco, United States: IEEE Computer Society, 2010, pp. 3304–3311.

[232] R. Hadsell, S. Chopra, and Y. LeCun. “Dimensionality Reduction by Learning an Invariant Mapping”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 2. June 2006, pp. 1735–1742.

[233] H. Oh Song et al. “Deep Metric Learning via Lifted Structured Feature Embedding”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[234] K. Sohn. "Improved Deep Metric Learning with Multi-class N-pair Loss Objective". In: *Advances in Neural Information Processing Systems*. Ed. by D. D. Lee et al. Curran Associates, Inc., 2016, pp. 1857–1865.

[235] M. Feng et al. "2D3D-Matchnet: Learning To Match Keypoints Across 2D Image And 3D Point Cloud". In: *2019 International Conference on Robotics and Automation (ICRA)*. May 2019, pp. 4790–4796.

[236] Y. Zhong. "Intrinsic shape signatures: A shape descriptor for 3D object recognition". In: *The IEEE International Conference on Computer Vision Workshops, ICCV Workshops*. 2009.

[237] A. H. Lang et al. "PointPillars: Fast encoders for object detection from point clouds". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 12697–12705.

[238] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[239] F. Schroff, D. Kalenichenko, and J. Philbin. "FaceNet: A unified embedding for face recognition and clustering". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015.

Acronyms

ADAS	Advanced Driver Assistance Systems
ANN	Artificial Neural Network
ASPP	Atrous Spatial Pyramid Pooling
BEV	Bird's-Eye View
BP	Back-Propagation
BPTT	Backpropagation Through Time
CNN	Convolutional Neural Network
CPT	Conditional Probability Table
DARPA	Defense Advanced Research Projects Agency
DNN	Deep Neural Network
EV	Embedding Vector
GAN	Generative Adversarial Network
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GPU	Graphics Processing Unit
HMM	Hidden Markov Model
LiDAR	Light Detection And Ranging
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MLP	Multilayer Perceptron
NLOS	non-line-of-sight
OG	Occupancy Grid

OSM OpenStreetMap

PnP Perspective-n-Point

RNN Recurrent Neural Network

RPN Region Proposal Network

ReLU Rectified Linear Unit

SLAM Simultaneous Localization And Mapping

SVM Support Vector Machine

SfM Structure from Motion