Spatial Road Representation for Driving in Complex Scenes by Interpretation of Traffic Behavior

Edoardo Casapietra

Dissertation

vorgelegt zur Erlangung des Grades
Doktoringenieur

Bielefeld University – Faculty of Technology
P.O. Box 10 01 31
D-33501 Bielefeld, Germany

Gedruckt auf alterungsbeständigem Papier °° ISO 9706

Edoardo Casapietra
CoR-Lab – Research Institute for Cognition and Robotics
Cognitive Robotics & Learning research group
Honda Research Institute Europe

Abstract

The detection of road layout and semantics is an important issue in modern Advanced Driving Assistance Systems (ADAS) and autonomous driving systems. In particular, trajectory planning algorithms need a road representation to operate on: this representation has to be spatial, as the system needs to know exactly on which areas it is safe to drive, so that they can safely plan fine maneuvers. Since typical trajectories are computed for timespans in the order of seconds, the spatial detection range needed for the road representation to achieve a stable and smooth trajectory is in the tenths to hundreds of meters. Direct detection, i.e. the usage of sensors that detect road area by direct observation (e.g. cameras or lasers), is often not sufficient to achieve this range, especially in inner-city, due to occlusions caused by various obstacles (e.g. buildings and high traffic) as well as hardware limitations. State-of-the-art systems cope with this problem by employing annotated road maps to complement direct detection. However, maps are expensive to make and not available on every road. Furthermore, ego-localization is a key issue in their usage.

This thesis presents a novel approach that creates a spatial road representation derived from both direct and indirect road detection, i.e. the detection and interpretation of other cues for the purpose of inferring the road area layout. Direct detection on monocular images is provided by RTDS, a feature-based detection system that provides road terrain confidence. Indirect detection is based on the interpretation of the other vehicles' behavior. Since our main assumption is that vehicles move on road area, we estimate their past and future movements to infer the road layout where we cannot see it directly. The estimation is carried out using a function that models the probability for each vehicle to traverse each patch of the representation, taking into account position, direction and speed of the vehicle, as well as the possibility of small past and future maneuvers. The behavior of each vehicle is used not only to infer the area where road is, but also to infer where there is not. In fact, observing a vehicle steering away from an area it was predicted to go can be interpreted as evidence that said area is not road.

The road confidences provided by RTDS and behavior interpretation are blended together by means of a visibility function that gives different weights to the two sources, according to the position of the patch in the field of view and possible occlusions that would prevent the camera to see the patch, thereby leading to unreliable results from RTDS.

The addition of indirect detection improves the spatial range of the representation. It also exploits the scenarios of high traffic that are the most challenging ones for direct detection systems, and allows for the inclusion of additional semantics, such as lanes and driving directions. Geometrical considerations are applied to the road layout, obtaining a distributed measure of road width and orientation. These values are used to segment the road, and each segment is then divided into multiple lanes based on its width and the average width of a lane. Finally, a driving direction is assigned to each lane by observing the behavior of the other vehicles on it.

The road representation is evaluated by comparison with a ground truth obtained from manually annotated real world images. As in most cases the entirety of road area cannot be seen in a single image (a problem that human users share with direct detection systems), every road is annotated in multiple different images, and the road portions observed are converted into BEV and fused together using GPS to form a comprehensive view of said road. This ground truth is then compared patch-wise to the representation obtained by our system, showing a clear improvement with respect to the representation obtained by RTDS alone.

In order to demonstrate the advantages of our approach in concrete applications, we set up a system that couples our road representation with a basic trajectory planner. The system reads real-world data, recorded by a mobile platform. The representation is computed at each frame of the stream. The trajectory planner receives the current state of the ego-car (position, direction and speed) and the location of a target area (from a navigational map), and finds the path that leads to the target area with minimum cost.

We show that indirect road detection complements direct detection in a way that leads to a substantial increase in spatial detection range and quality of the internal road representation, thereby improving the smoothness of trajectories that planners can compute, as well as their robustness over time, since the road layout in the representation does not dramatically change only when a new road is visible. This result can help autonomous driving systems to achieve a more human-like behavior, as their improved road awareness allows them to plan ahead, including areas they do not see yet, just as humans normally do.

Acknowledgments

I would like to thank the CoR-Lab and the Honda Research Institute Europe for giving me the opportunity and the means to work on this project.
Likewise, I would like to thank all the people that helped me along the way and made this project possible, in particular Dr. Thomas Weisswange, for his constant input and feedback, Prof. Franz Kummert and Prof. Christian Goerick.
Finally, I want to mention all the members of HRI-EU, and especially my fellow students there, for the great time we spent together.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Chapter overview *This chapter introduces the research topics presented in this work, and establishes some basic terminology. The chapter closes with an overview of the key contributions and structure of the thesis.*

1.1 Advanced Driving Assistance Systems

In the past century the car industry focused on improving crash safety by developing passive systems, such as seat belts, ABS, Air Bags, etc. Recent technological advancements (especially in electronics and informatics) allowed to move the main attention to "intelligent" active systems, which are called Advanced Driving Assistance Systems (ADAS). These systems have two main purposes, first and foremost, to increase safety by preventing dangerous situations and road casualties. In fact, cars are the means of transport that produce the most fatalities in the world, even though the figures have significantly decreased (especially in the western world) over the last 50 years (Jacobs & Aeron-Thomas, 2000), thanks to the aforementioned systems. The second purpose is to increase the comfort for the driver, especially in common situations, where driving is tedious or annoying, like in traffic or on the daily commute. This can be achieved by letting the system take partial control of the vehicle and allow the driver to spend their time freely during the ride.

In order to be able to take over driving tasks, these systems need to have an understanding of the scene they operate on. When they are on the road, they need to detect and recognize the elements that are present or act in their environment, such as other vehicles, pedestrians, buildings, obstacles, curbs, and the road itself (see figure 1.1 as an example). The amount of awareness required for driving assistance systems depends on the complexity of the task, and has therefore increased in recent years. For example, an ABS only needs data about the ego-car itself to be effective. Such data can be collected by simple sensors on the wheels. The system also needs actuators to correct the breaking of the car, in case the wheels lock. A more complex system is ACC (Advanced Cruise

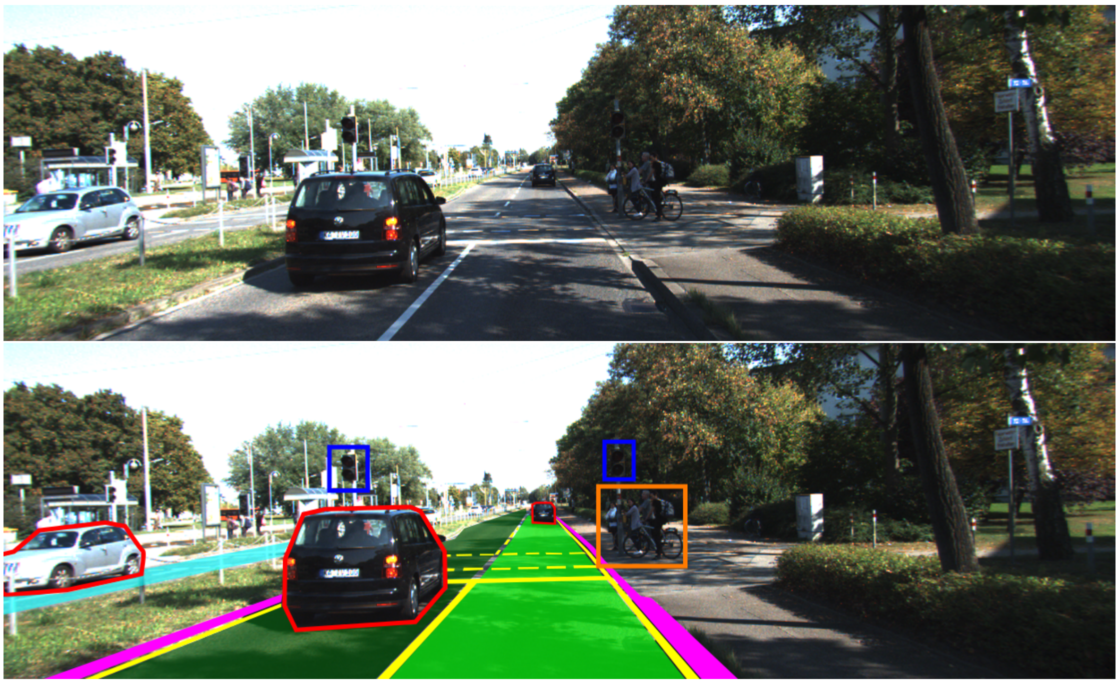


Figure 1.1: Above: A typical road scene as seen by a camera. Below: Highlight of the elements that an ADAS might need to detect; the ego-lane (light green), the other lane (dark green), opposite lanes (light blue), lane markings (yellow), curbstones (violet), other vehicles (red), pedestrians (orange), and traffic lights (dark blue).

Control), whose function is to manage the speed of the ego-car in a way that keeps the distance to the vehicle in front. It detects the car directly in front of it, its position and its speed (usually done by radar or camera), and has to be aware of the speed of the ego-car itself. The system needs a simple logic to react to variations in speed of the object in front (it does not even need to recognize it as a car) in order to adjust the ego-car speed accordingly. Another common ADAS is a lane keeping system, that makes sure that the ego-car does not leave its lane. This system needs to detect road and be able to segment it based on e.g. lane markings (that also need to be detected and recognized), for which a camera is commonly used. It must be aware of the direction of motion of the ego-car and its dynamics. It also needs to know how to safely steer the ego-car, and have actuators able to do so.

If we want to progress further, we should shift the focus from correcting a risky behavior to preventing it. This is in fact the long term goal of ADAS research. In order to accomplish it, an ADAS needs to be able to predict future situations, by reasoning on the current and past scenes. For example, consider a predictive braking system, which predicts a possible collision a few seconds in advance and reduces the speed of the ego-car safely to prevent it. It will need to be aware of all of the elements mentioned above, with the addition of all the road actors in the vicinity, and it will need to understand relations between those elements in the scene. The detected cars will need to be related to their lanes, in order to predict their motion and to understand if they represent a danger for the ego-vehicle. Additionally, such a system would need an extended detection range, because the events that it has to predict would depend on entities or objects that may be far away, since they can move at significant speed and cover the distance in those few seconds. As a further step, such a system would need to have a driver behavior model to understand the maneuvers made by the humans in the scene. If we want to use this system in inner city, we also want it to warn us about pedestrians potentially crossing the road. In this case it would need to be able to not only detect the pedestrians and correctly classify them (i.e. understanding that the detected objects are pedestrians), but also recognize that a pedestrian is likely about to cross the road in front of the vehicle. It might also be necessary to communicate its reasoning to the driver via an HMI. These kind of ADAS will be a step further towards autonomous driving systems. Fully autonomous driving will need to manage the ego-car on the short term, by predicting immediate risks and preventing them, and on the long term, by planning routes and trajectories to reach the destination. Performing these tasks will require all of the resources already mentioned, plus others, such as for example navigational maps.

Roads are an extremely dynamic environment. For this reason it is very important for ADAS to detect its elements as early as possible: early detection can help to predict the trajectory of the other vehicles, to evaluate risks associated to other traffic objects and to plan longer and smoother trajectories for the ego-vehicle itself. Regarding ego-motion, we can distinguish three levels of planning, each acting on a different time scale and thus requiring a different detection time horizon. The lowest level is the reaction to unexpected events, where a system has to plan short term maneuvers and corrections. It has to deal with a very short time horizon (max. 1-2 seconds) and requires high spatial accuracy, that is necessary to safely avoid obstacles or correct trajectories that were computed previously with a lower spatial accuracy. This level is managed using a wide array of sensors (e.g. lasers, cameras, radar), that detect road and objects by sensing them directly with high accuracy but limited detection range in terms of distance. For example, emergency breaking systems have to operate at this level, since their purpose is to detect close objects with high reliability. Cruise control and lane keeping systems also operate mainly at this scale, as they just need to react to changes in speed of the preceding vehicle and to changes in direction of the lane itself. The highest level is route planning, whose goal is to pick the most efficient roads in a network to reach a given destination. It has to deal with a time horizon of minutes or even hours, but it does not require high spatial accuracy. In fact, the most important information at this level is the topology of the roads and other properties, like the number of lanes, their direction, their speed limit, their average traffic, and so on. Modern ADAS cope with this level by using offline navigational maps, that provide knowledge of the road ahead for a virtually unlimited time horizon, and the information they provide is exactly the kind that is needed for this level of planning. Finally, there is a mid-level between the two above. It is the one that plans trajectories, and deals with time horizons of a few seconds. This level needs spatial knowledge of the road, as it has to make sure the planned trajectory stays on it, but it also needs higher-level information (for example, it needs to compute the short-term destination of the maneuver). State-of-the art control systems cope with this level of planning by using a mix of all the resources mentioned above (Bacha et al., 2008)(Miller et al., 2008)(Urmson et al., 2008). The result, however, is far from optimal, since on one hand the conversion of map information from topological to spatial is not trivial and causes high uncertainty, and on the other hand the accuracy of direct road detection is not as good as desired at the spatial horizon needed for this time scale.

The understanding of the environment that a system needs is often achieved by building an internal representation of the environment, upon which the system can make its computations. Environment representations can have very different forms, depending on the purpose they serve. They differ in *what* they represent and in *how* they represent it. The traffic environment has many aspects that a system might need to represent. There is the road area itself, but there are also curbs, pavements, traffic signs, as well as other road actors, e.g. vehicles and pedestrians. Even an apparently simple entity like road can be represented in various ways: if the purpose is route planning, road is mostly represented as a topology of road segments, as the representation is needed purely to choose the most efficient sequence of roads, which is a problem that is commonly solved by using graphs. If the purpose is obstacle avoidance, roads need to be represented spatially, as the system must know exactly the spatial relation between the obstacle and the ego-car in order to both assess the risk and react accordingly. In the latter case, the representation can treat space as a discrete entity, by using a grid-based format, or treat it as continuous, by using analytical functions, e.g. spline curves. A representation can also focus on different aspects of road terrain. It might target only free space, which is the part of the road ahead that is not occupied by obstacles or other actors. Conversely, it might target the whole road area, regardless of its availability. Lastly, it might even target any road-like area, that is any drivable terrain that is or is not formally part of the road, like parking spots. All these different varieties of "road" can be part of the scope of a representation, depending on the purpose of the system that uses it. Additionally, road area can also have multiple semantics. It can usually be divided in multiple lanes, each with a driving direction assigned. Certain roads, or parts of them, can be forbidden (e.g. emergency lanes in highways), or can have different speed limits. This additional information can be useful for an ADAS and should definitely be available for autonomous driving vehicles.

The data used to build a representation can be provided by external sources (e.g. annotated maps), or taken on site by means of sensors. A typical issue of many modern ADAS that has been addressed only lately, is that if different systems are on the same car, they often use different sensors and separate representations. This approach is not optimal, as fusing information between different sources can lead to a better understanding of the scene. Multiple ADAS should share their internal representation, and build it using all the available resources. Furthermore, it has to be noted that sensors are a limited resource: they can be expensive and they occupy valuable space in the car. Car manufacturers need to consider these factors when deciding which and how many sensors their cars should be equipped with. From this perspective, it is clear that having redundant sensors is desirable on one hand, as it provides more reliable information, but undesirable on the other, as it can be expensive and cumbersome. Therefore, ADAS should exploit all available information to the fullest of its potential, and that means also exploring new ways to extract additional information from the same sensor signals. This is one of the reasons that leads to the concept of *indirect detection*, that we define as inferring information by reasoning upon related entities that have been detected. This concept is often used by human drivers: if a vehicle in front of us brakes just before a zebra crossing, we can assume that there are pedestrians crossing the road, even if we cannot see them as they are occluded by the vehicle itself. Indirect detection allows us to detect entities using sensors that are meant to detect something else, or to expand the detection range of our sensors, detecting entities that are too far away by reasoning upon closer entities that are easier to detect. The ability to use different resources to detect the same entity, and at the same time to detect more entities with the same resource, increases the overall detection range of the system and allows for a certain degree of redundancy without the need of additional sensors, thereby increasing the reliability of the internal representation.

1.2 Motivation

As mentioned in the previous section, planning at a mid-level time horizon is a challenge for autonomous driving systems, as neither of the resources used for the other two levels (direct detection and navigational maps) is optimal for it. The insufficient spatial accuracy of the direct detection methods at a mid-level time horizon is due to various reasons. While the ego-vehicle is moving, a time horizon of a few seconds translates to a spatial range of tenths of meters. The longer the range is, the lower the accuracy. Firstly, every sensor has hardware limits (e.g. camera resolution) and their performances get naturally worse with distance. Secondly, the nature of road environment itself, where occlusions are very frequent and are a key aspect of the scenario, provides considerable challenges to all direct detection systems: the traffic can hide large parts of road from our field of view (FOV), and the same is true for buildings and other obstacles, in particular at intersections and curves. The parts of road that are hidden by occlusions get larger with the distance, so that the farther away the road we look at is, the likelier it is that our sensors cannot sense it at all. It is important to note that all the problems mentioned above are most frequent in inner city environments rather than in more

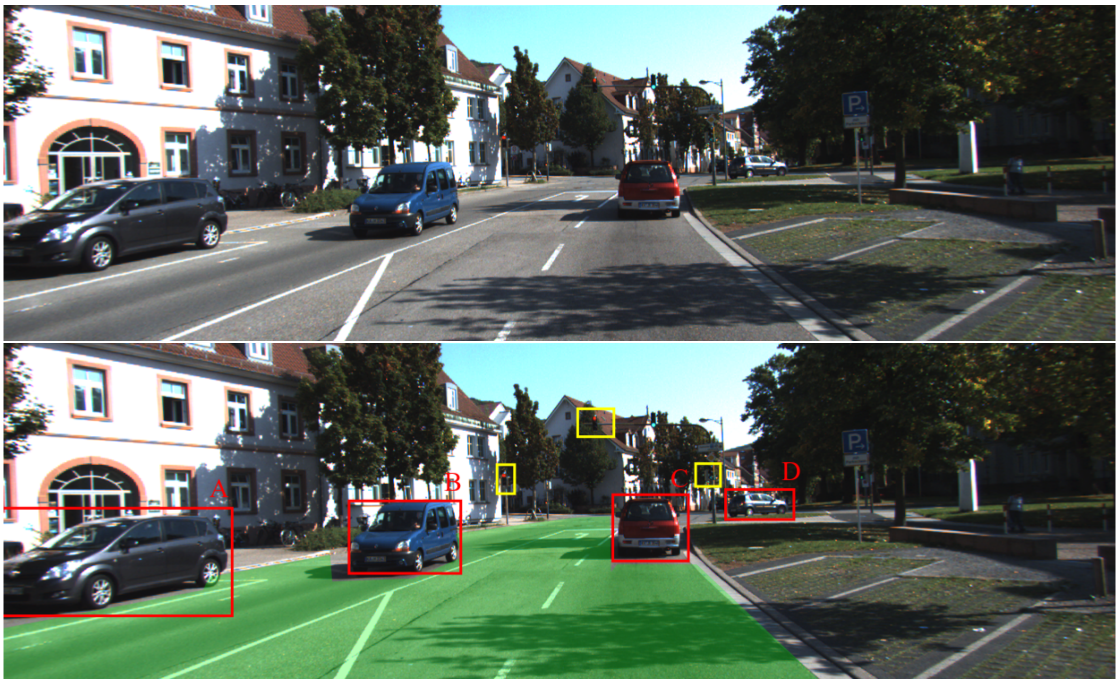


Figure 1.2: Typical urban intersection as seen by a camera (above). Same scene after road and object detection (below).

simple environments like highways, which is one of the reasons why commercial ADAS still struggle to perform in inner city as reliably as they can currently do on highways.

Humans deal with occlusions in multiple ways. In most cases, we can understand the layout of a road even if we cannot see it. If we see a scene like the one in figure 1.2 we immediately recognize that the sudden enlargement of the road ahead is just part of an intersection, and that we can expect there to be roads on both sides. Our understanding comes from our own experience: we know how a typical road intersection looks like, and the traffic lights, the lane markings and even the disposition of the buildings on the left are all strong evidence of the presence of an intersection. This kind of reasoning however is not trivial for an AI that bases its understanding on detecting directly the road. The intersection itself can barely be seen at all, because most of it is occluded by the car in front of us, and the part that is visible to us is far ahead, at a distance where most direct detection method struggle with. Even our own eyes, that have much better performances than state-of-the-art computer vision methods at this range, can just get a glimpse of the road going to the right, even if said road is not occluded at all. However, we can be fairly confident that there is indeed a road there, because we can see that car D has just turned right at the intersection. This reasoning is not limited to road area only. Even if we cannot properly see the traffic light but vehicle C in front of us stops, we can assume that it is red. If we can see that the traffic light is green but the vehicle still brakes, we can assume that there could be an obstacles on the road in front

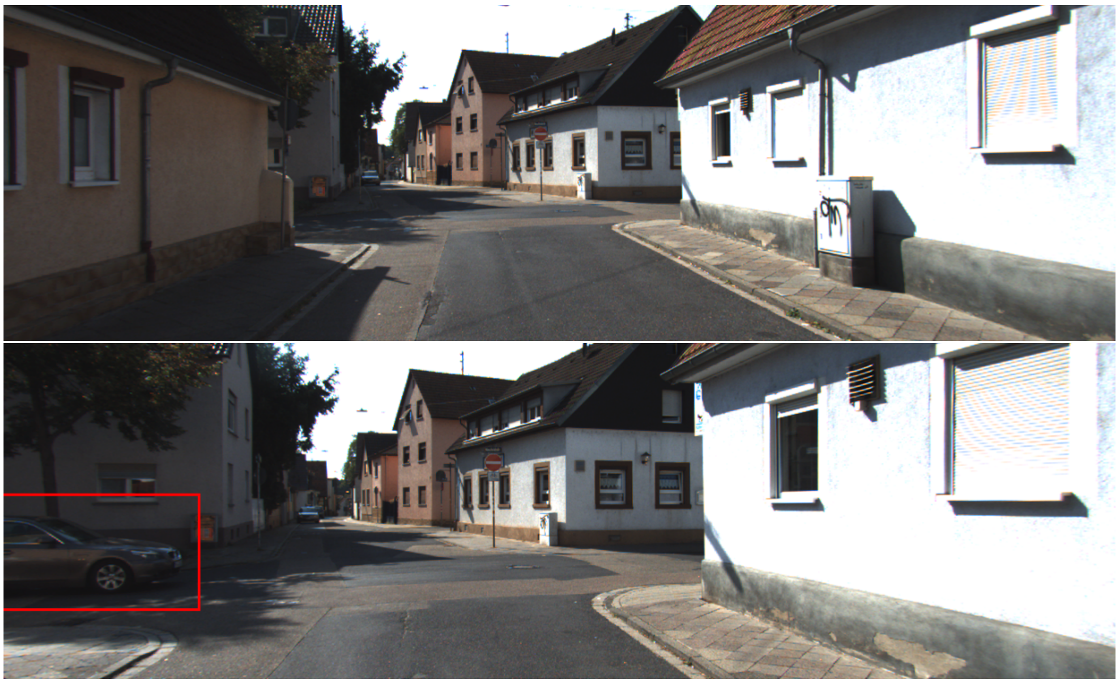


Figure 1.3: Urban intersection in two different instants. The appearance of the car from the left side allows us to infer the presence of a road there.

of it. As a further example, figure 1.3 shows a typical inner city intersection. We as humans can easily recognize it as such due to our driving experience, but an autonomous system would not be able to notice the two side roads and only detect the one in front of us, due to the buildings occluding most of the scene. In fact, the large shadows on the left side and the surface difference on the road itself make it a significant challenge for any autonomous system to properly detect even the non-occluded road. However, at some point the intersection is traversed by a car, coming from the left and going right. We normally interpret the appearance of that car in two ways: firstly, the car must be coming from an occluded road on the left. Secondly, the car is likely heading towards an occluded road on the right. The behavior of the car becomes then evidence that there is indeed a drivable road on both sides of the intersection, but this happens only because we are interpreting it. We refer to this concept as ”Behavior Interpretation” in this thesis.

As explained above, we can exploit the fact that we are not alone on the road, and an AI should be able to do it as well. In fact, every other driver sees a different scene from the one we do, and has therefore a slightly different understanding of the environment, since they can sense different portions of the environment. For example, a car driving on one of the side roads can see its road but cannot see ours, or a car that is coming from our opposite direction cannot see what lies behind us but does know the road it just went through, while we cannot see it yet. Hence, although every vehicle on the road can potentially occlude our field of view and reduce the information we can gather directly, that information is not lost, because it is retained in the environment representation of those other vehicles (be it that of human drivers or AIs). This concept is one of the main motivations of all the systems that use or plan to use car-to-car communication: the idea is that all vehicles in the area would be able to share their representation and fuse it into a full understanding of the environment. However, this approach presents numerous challenges. First of all, it requires multiple cars to have the same system installed, or at least multiple compatible systems. Furthermore, establishing a communication protocol between multiple cars is not a trivial task, and it would probably require to set up ad hoc infrastructures on the road to serve as communication hubs. Finally, data-security and privacy is an issue that needs to be taken into account. For this reason we want to focus our efforts on how to retrieve at least part of the information held by the other vehicles as an independent system. Our problem resembles a classical system's theory problem: we have a black box (an unknown vehicle) that has an internal state (that includes its environment representation) that we want to estimate from outside. Exactly as it happens for usual systems, the vehicle has an output that we can observe: its behavior. In fact, the behavior of any car depends on its representation, just as the output of a dynamical system depends on its state, although the nature of this dependency, as well as the shape of the output, are not as straightforward as in the latter case. Following this reasoning, a possible way to retrieve some of the lost information and estimate their internal representation is by observing and interpreting their behavior. Unfortunately, their full representation cannot be recovered, since their behavior is affected only by part of it, but the recovered part might still be significant for us. In fact, since it is inferred from the motion of vehicles, it is reasonable to think that such information could be especially significant for the motion of our vehicle too. Following this reasoning, we expect the retrieved information to be relevant for planning the motion of the ego-car.

1.3 Aim of the Thesis

In this thesis we present a novel probabilistic framework for a 2D grid-based spatial road representation, which uses a novel indirect detection approach, based on the considerations discussed in the previous section, coupled with an already existing direct detection method. The representation consists of a regular square grid, where every square (which we call "patch") holds a probability of being part of road area. The indirect detection uses the motion of other vehicles in a scene as a cue to infer road probability, and it is based upon the position and velocity of the vehicles in the scene, features that at mid-range can be estimated with more reliability than road area itself (Petrovskaya & Thrun, 2009). As a direct detection algorithm, we use the SPRAY-based visual road terrain detection system (RTDS) published in (Fritsch, Kühnl, & Kummert, 2014). We chose this system for its good performances in the KITTI benchmark (Geiger, Lenz, &

Urtasun, 2012), but any kind of direct detection system providing spatial road confidence values can be used in this framework. The two sources (direct and indirect) are fused together patch-wise, based on consideration on the reliability of direct detection and potential occlusions detected.

Since it exploits the vehicles in the scene, the indirect detection approach shows the biggest advantages in those situations that are most challenging for direct detection systems, and vice-versa. Therefore, fusing the two sources together allows our system to minimize the flaws of both. The framework can be expanded by adding semantics to the road itself. In particular, we show how to infer the lane layout and the driving direction of each lane by geometrical considerations and by further reasoning upon the behavior of the other vehicles.

In order to evaluate the improvement in road detection provided by the addition of our indirect approach, we compare the performances of our full framework against those of a baseline system featuring direct detection only. The performance is computed on real-world recordings based on a manually annotated ground truth. Furthermore, we set up a system that uses our framework and couples it with a trajectory planner, and compare the trajectories obtained by the full system against those obtained with the baseline system. The planner builds a tree of spatial nodes and uses a tree-search algorithm to find the path with the minimum cost that leads to a target area defined by the user. The evaluations show that our approach improves significantly the road detection with respect to the baseline performances. It can enhance the detection of roads in intersections, and can detect side roads that lie behind occlusions, invisible to any direct detection system. Furthermore, it allows to compute trajectories that are more spatially accurate and more stable over time compared to the ones obtained with the baseline representation.

1.4 Outline

The remainder of this thesis is organized as follows: chapter 2 will present the literature related to the topic. The following chapters will detail the theoretical framework for road representation that we designed. Chapter 3 will focus on road probability estimation, while chapter 4 will deal with lanes and their driving directions. Chapter 5 will present the concrete system that we implemented to test our framework, and chapter 6 will show and discuss the experiments we ran on it. Finally, chapter 7 will wrap up the thesis with a discussion about the results of the thesis and future work.

Chapter 2

Related Work

Chapter overview *This chapter provides an overview of the state-of-the-art of autonomous driving, in particular in the three topics that are directly relevant to this thesis: how to perform road detection, how to represent the road environment, and how to plan trajectories on it.*

Current state-of-the-art autonomous vehicles employ accurately annotated lane-level maps to obtain road environment information. For these systems, the main problem to solve is self-localization and pose estimation (Matthaei, Bagschik, & Maurer, 2014). In fact, the vehicles competing in the DARPA Urban Challenge 2007 carried multiple LIDARs, RADARs, as well as high-end GPS and IMU sensors, but were also provided with a detailed digital map of the road network, including updated aerial images (Bar Hillel, Lerner, Levi, & Raz, 2014). The main purpose of the sensors was to localize the ego-car within the maps, and to detect obstacles and other traffic participants.

In 2013, Daimler performed the BERTHA experiment (Franke et al., 2013). BERTHA was a Mercedes S-Class with close-to-production sensors that drove autonomously along the 100 km long Bertha Benz Memorial Route, between the German cities of Mannheim and Pforzheim. This experiment also relied on accurately annotated digital maps, and used cameras, radar and high precision GPS sensors to localize itself and to match the sensed environment to the expected one. Although this approach did work well, it is not feasible for commercial systems to rely only on annotated maps and high quality localization systems, as it would be too expensive and not flexible enough. Onboard environment perception performed with affordable sensors would be a more viable option. Thus, the main problem is to reliably and accurately detect the environment, and to build an internal representation of it.

2.1 Road Detection

Road detection is a key issue for modern ADAS. The concept of *Road* has multiple aspects. In particular, it has a physical aspect (physical boundaries, e.g. curbs, and a drivable surface) and a regulatory aspect (formal boundaries and areas, e.g. road markings, emergency lanes, etc.). An ADAS may need to focus on both aspects, especially for autonomous driving purposes, where the aim is to understand not only where the car *can* drive, but also where it *should* drive. Over the years, different classes of methods have been developed with the aim of making road detection as accurate and reliable as possible. We can distinguish two main classes of road detection: direct and indirect methods. Direct methods rely on sensing the road directly, e.g. a camera actually seeing the road, or a laser hitting the road. Conversely, indirect methods detect road by sensing something else, e.g. other vehicles.

2.1.1 Direct Detection Methods

Direct road detection algorithms can be divided into two categories, marked and unmarked. Marked road detection infers road area by detecting lane markings and applying prior knowledge about road geometry (Weigel, Cramer, Wanielik, Polychronopoulos, & Saroldi, 2006). It is most commonly used in simple and highly structured environments like highways. Unmarked road detection systems instead can infer road by observing low-level visual features, such as color, texture and elevation.

Marked road detection usually aims to model lane markings as a spline curve, in order to obtain a solid lane or road boundary from detecting part of the lane marking itself, and to help the detection in subsequent frames. For example, (Wang, Shen, & Teoh, 2000) proposes interpolating lane markings using Catmull-Rom splines, which hold several good properties. They allow for arbitrary shapes, they generate continuous and smooth curves, and they are locally controllable, i.e. small changes in a control point position cause changes in shape that are limited to the surroundings of that point. Furthermore, their two coordinates x and y can be decoupled, allowing for a different parametric representation of the two. The approach presented is visual-based and involves several steps. First, the vanishing line vertical coordinate is estimated, from which they start extracting control points (detected by a simple edge detection filter). The control points are determined by employing a lane model to compute the likelihood that each edge point belongs to a possible lane marking. After the extraction is completed, the spline curves are finally computed. (Jung & Kelber, 2004) proposes a different lane model, where images are split into a near- and a far-field through a horizontal threshold. In near-field, lane markings are modeled with a linear function, while in far-field they are approximated with a parabolic function. This approach is simpler but provides a model that is accurate enough for many applications.

Unmarked road detection is usually performed by using pixel-level feature-based classification. It is often based on camera images, where features can be computed from color or texture, but it can also be achieved by using other sensors. For example, (Caltagirone, Scheidegger, Svensson, & Wahde, 2017) employs a fully convolutional neural network to detect road at pixel level by using only LIDAR-based point cloud. From this point cloud, the approach builds a top-view representation encoding six basic statistics on each 0.1

x 0.1 cell. The statistics used are: number of points, mean reflectivity, and mean, standard deviation, minimum and maximum elevation. Six top-view images are generated, encoding each statistic, and fed as inputs to the network, which outputs a road confidence map. The approach, due to its usage of only LIDAR data, was independent from lighting conditions and could work in real time, giving state-of-the-art performances.

Researchers have also explored the possibility to realize hybrid approaches by combining low-level features with contextual cues and priors knowledge. In this direction, (Alvarez, Lopez, Gevers, & Lumbreras, 2014) presented an approach to estimate road priors (e.g. road type, driving direction, number of lanes) from geographical information (by using GPS data and a road database). Subsequently, they extract a set of low-level features (e.g. color) and a set of five contextual cues. These were horizon line height, vanishing points location, lane markings, 3D scene layout (i.e. the segmentation between sky, vertical surface and ground pixels), and road shape (classified using a Support Vector Machine trained on multiple images). Finally, they use a generative model to combine all these cues and priors, obtaining a road detection method that is robust to varying scenarios. Another work that exploits prior knowledge is (Brust, Sickert, Simon, Rodner, & Denzler, 2015), which proposed an image-based convolutional neural network that detects road (and other semantic entities such as buildings, pavements, cars, etc.) by using visual features together with the normalized position of each patch within the image. This approach arises from the observation that the position of different categories is not uniformly distributed in the typical image (e.g. road is usually located in the lower part, the sky is at the top, etc.). The inclusion of position information led to a significant performance improvement over other state-of-the-art methods.

The road detection system that we use in this work is the Road Terrain Detection System (RTDS) presented in (Fritsch et al., 2014). This method consists of two stages. The first stage includes three base classifiers, which extract local visual appearance properties, such as color and texture. A base road classifier is specialized in generating high confidences on road-like area and low confidences on non-road terrain. A base boundary classifier detects boundaries between road-like area and adjacent regions, e.g. sidewalks, traffic islands, turf. A lane marking classifier detects lane markings by employing a dark-light-dark filter. All these three classifiers output confidence maps that are fed to the second stage. The second stage analyzes spatial ray features (SPRAY) that are computed on a set of uniformly distributed base points from the confidence maps obtained by the three base classifiers. These features are based on lines (rays) starting from each base point along a fixed set of R directions. Along each ray the system computes the integral of the confidence values (which they call "absorption"). A set of thresholds T is defined, and the distance ρ at which the absorption hits each threshold is taken as a feature. The process is executed for each base points, so that at the end of the feature generation each base point has an $R \times T$ feature vector. Additionally to these "individual" SPRAY features, another set of features, "ego"-SPRAY features, is computed.

This kind of feature uses the same approach, but applies it to the lines that connect each base point to the ego-car position. Finally, another classifier performs road terrain classification using the set of SPRAY features obtained above as inputs.

The need of an evaluation benchmark for direct detection methods led to the development of KITTI (Geiger et al., 2012), a public computer vision benchmark for the tasks of optical flow, visual odometry and 3D object detection in real-world road environment. KITTI was recorded using a mobile platform provided with a frontal monochrome and stereo camera, a 360 deg Velodyne laser scanner and a GPS localization system. The scenes were recorded in the city of Karlsruhe, Germany. KITTI has quickly become the most popular benchmark dataset for road detection, but since it was created with direct detection in mind, the recorded scene are not well suited to evaluate indirect detection approaches.

2.1.2 Indirect Detection Methods

Due to the challenges that direct detection methods face in complex environments, researchers have started developing alternative methods to infer the environment layout without sensing it directly, which are what we call indirect detection methods. The most promising indirect cue that has been investigated is the behavior of other entities in the environment. This is because other vehicles are usually easier to detect than road itself or other visual cues, and their behavior is intrinsically related to the environment layout. Initially, the behavior of vehicles was the target to be inferred, and the road (or the environment in general) was the cue, often helped by prior assumptions about the generic behavior of vehicles in traffic. For example, behavioral assumptions have been exploited to improve the tracking and predicting the behavior of vehicles (Alin, Butz, & Fritsch, 2011). This work introduced a grid-based population code based on a spatially distributed hidden Markov model. The approach used lane information, together with the assumption that vehicles tend to stay on their lane, to estimate and predict current and future position of a vehicle during overtaking maneuvers in a simulated environment.

Behavioral assumptions have also been used to predict vehicles motion at intersections (Liebner, Baumann, Klanner, & Stiller, 2012). Here, the authors employ an Intelligent Driver Model to estimate the observed vehicle intent by its speed profile approaching the intersection. The model has been developed to take into account the possible presence of a preceding vehicle, that has an influence on the speed profile of the analyzed vehicle. The approach could discriminate between three behaviors: go straight, turn right, stop before turning right. The geometry and location of the intersection had to be known beforehand for the approach to work.

Information about context can be also used to classify the related danger of pedestrians or vehicles. For example, (Weisswange, Bolder, Fritsch, Hasler, & Goerick, 2013) focused on the assessment of risky behavior from detected vehicles in the scene. This approach classifies all detected vehicles as risky or not risky for the ego-vehicle, by estimating their position, speed and orientation with respect to the ego-car and to the detected road area. In order to evaluate the risk associated with each vehicle, they employ a decision tree to analyze the situation, which takes into account different estimated values such as relative position and time to contact between ego-car and vehicle, type and orientation of the vehicle and its distance to road area if it is outside of it.

(Bonnin, Weisswange, Kummert, & Schmüdderich, 2014) pointed out that the best inference quality is obtained with narrow-scope approaches, and thus proposed a general context-based approach, that combines multiple classifiers for different scopes (broad and narrow), and selects them based on the situation, employing a scenario model tree (SMT). The SMT manages the various classifiers, and handles dependencies (every node depends on its parent, which is a broader-scope classifier) and competing classifiers: if a situation includes multiple classifiers, a competition function modulates the results of each classifier to obtain a single response.

While these works focused on inferring behavior by using context, the approach we present in this thesis aims to do the opposite: exploiting the behavior of vehicles to infer the context. In this direction, the behavior of pedestrians has been used to infer navigational maps for robots (O'Callaghan, Singh, Alempijevic, & Ramos, 2011). In this work, the authors track pedestrian motion and compare it to a prior navigational map, that indicates the motion direction on any point towards the goal. Although such map could be derived from a variety of sources, they use a naive version, that simply assigns to each location in the map a direction pointing straight to the goal. The pedestrian motion direction is compared to the prior direction point by point, and the difference is then fed as a data point to a Gaussian process that estimates a "deviation" function, which expresses the local difference between prior and actual pedestrian motion. As the approach depends on the goal location, only the motion of observed pedestrians actually going through the goal was considered. The deviation function is finally used together with the prior map to guide a simple trajectory planner, which follows the direction provided by the combination of the two sources all the way to the goal.

The behavior of vehicles has been used, together with lane markings detection and context models, to infer road geometry at long range on highways (Fatemi, Hammarstrand, Svensson, & García-Fernández, 2014): this work used a clothoid model to approximate the road shape, and used the heading of detected vehicles and lane markings to estimate the parameters of the clothoid. For this purpose, they had to take into account two vehicle behavior models: one where the vehicle drives parallel to its lane, and one where the vehicle is exiting or entering the lane. The transition between the two models was managed by means of a transitional probability matrix. The experiments showed that the addition of vehicle measurements decreased the error at long range with respect to the sole usage of lane markings. Similarly, the behavior of vehicles has also been used together with other visual cues in order to infer the topological and geometrical layout of road intersections. (Geiger, Lauer, Wojek, Stiller, & Urtasun, 2013) achieves this goal by using multiple cues: vehicles tracklets, vanishing points, semantic scene labels, scene flow and an occupancy grid. These cues allow to infer a set of road properties, including the topology of the intersection, predefined as a set of seven possibilities, the location of the center, the rotation of the intersection with respect to the frame of reference, and the crossing angle, i.e. the relative orientation of the crossing street with respect to the incoming street. The approach assumed that all roads had the same width, and it used a probabilistic generative model.

Vehicles behavior has been used not only to infer the road layout, but also the lanes. Guo et al. (Guo, Meguro, Yamaguchi, Kidono, & Kojima, 2014) use the past trajectories of other vehicles to refine the parameters of a clothoid modeling lateral lane boundaries, improving the robustness of lane detection in challenging scenarios. Thomas et al. (Thomas, Stiens, Rauch, & Rojas, 2015) combine direct lane detection with the trajectory of vehicles in the scene, which are used as a cue to for the lane center line. The resulting representation is a probabilistic spatial grid, where semantic functions created from different cues (lane dividers, road boundaries, dynamic objects) are merged together to obtain the probability of being part of the center line for each square of the grid.

However, all of these systems used only the trajectory (current and past behavior) of vehicles, while the approach we present in the following chapters exploits also the predicted behavior. Furthermore, these approaches tend to exploit behavior to infer parameters of a model, while we want to use them to directly infer road area on each location.

2.2 Environment Representation

Autonomous cars need an internal spatial representation of the environment to assess risks or plan maneuvers. As mentioned, the representation used by BERTHA was a set of pre-annotated digital maps, that were built to include all the road elements that the car might have had issues detecting (Bender, Ziegler, & Stiller, 2014). One of those elements was the layout of drivable lanes, which is especially difficult to detect at intersections. For this reason, BERTHA employed the notion of ”Lanelets”: atomic, interconnected lane segments with one entry and one exit. Lanelets represented both topological and geometrical aspects of lanes. This detailed representation was built in a semi-automated manner based on images from stereo-camera and a very accurate (and expensive) DGPS, which was not required for the demonstration run itself, but only for the preliminary map-building process. It is clear that such a representation, while very accurate and reliable, needs a precise an extensive knowledge of the environment, which at the moment is too expensive and resource-consuming to be available for large-scale commercial purposes.

For many road-related applications, a two-dimensional representation is considered sufficient. These representations are usually discretized along both axis, by defining a 2D grid. The representation that we present in this thesis is also based on a regular 2D grid.

A very popular type of grid-based representation is the occupancy grid, a spatial representation that describes the surroundings of the ego-vehicle in terms of free or occupied space probability. Formally, occupancy grids are defined as two-dimensional arrays which model occupancy evidence of the environment, where the 3D world is orthogonally projected onto a plane parallel to the road. The plane is discretized into tetragonal cells that never overlap, which hold an occupancy likelihood of the represented area. In (Badino, Franke, & Mester, 2007) three main formats of camera-based occupancy grid are mentioned. The first and most common is the Cartesian map, which represent a portion of the surrounding environment with a constant resolution along the axis x (lateral) and z (depth). The second type is the Column-Disparity map, where the x axis is replaced by the lateral axis u of the camera, and the z axis is replaced by the disparity value d. The last type is the Polar occupancy grid, which has the same u lateral axis, but has z as depth axis. While a Cartesian grid provides an intuitive way to represent the environment, it has the drawback of its computational time, and the uneven relation between the resolutions of images and representation: far away pixels affect many more cells than closer pixels do. A Column-disparity grid is much faster to compute, and its lateral resolution is equal to the lateral resolution of the images. However, its depth resolution (tied to disparity) decreases quadratically with distance, so for applications that require high resolution at long distance the paper recommends a polar occupancy grid, whose depth resolution is constant. Unfortunately, polar maps are very complicated to manage over time, if the ego-vehicle moves (which is the situation to be expected for road applications). They are better suited for instantaneous applications, where information does not have to be carried out over successive frames, and a new occupancy grid is created at every iteration.

Since most grid-based representations are centered around the ego-car, its motion need to be taken into account, in order to create a correspondence between cells in different iterations. As the ego-car moves, a mechanism to shift the representation with it is necessary. As an example, (Weiss, Schiele, & Dietmayer, 2007) presented an online occupancy grid for estimating the driving path. The occupancy is computed from laser data, and the map is updated over time to represent a fixed area around the ego-vehicle. As the ego-vehicle moves, cells are created and eliminated at two ends of the grid. To avoid discretization errors while rotating cells, the grid orientation does not follow the ego-vehicle, but stays fixed, whereas the ego-vehicle can rotate with respect to the grid. The driving path is estimated starting from a center line, stemming from the ego-vehicle in the direction of its z-axis. On both sides of the center line multiple short sub-lines are created, parallel to it. The sub-lines move away from the center line until they find an occupied area. Finally, the sub-lines are interpolated to find the likely driving corridor boundary.

The classical formulation of occupancy grid, that was born for aiding robot navigation in closed environments, assumes a static environment. The presence of dynamic objects (e.g. other vehicles), which are very common in road environments, can raise issues, in terms of undesired artifacts in the representation. This is the reason why many recent approaches focus on tracking dynamic objects, sometimes treating them separately from the static occupancy grid. For example, (Gindele, Brechtel, Schröder, & Dillmann, 2009) proposed an occupancy grid where occupancy is preserved. It means that the representation tends to treat the amount of occupied cells as constant, and just moves occupancy around, based on the estimated velocity of detected objects. The representation is enhanced by a priori map knowledge, that helps to estimate and predict the motion of objects in the road environment. Map knowledge is exploited by using a reachability matrix $R_{a,c}$, which expresses the likelihood that an object on cell a could move to cell c. This matrix is computed by assigning a terrain type to each cell ("lane", "sidewalk" and "unknown"), and by following certain assumptions: objects tend to stay on their terrain type, and if they are on a lane, they follow its direction. Another interesting approach was presented in (Bouzouraa & Hofmann, 2010), an occupancy grid with focus on detecting and tracking moving objects. Their approach is based on laser and radar detection. Additionally to the usual occupancy probability, each cell in this representation holds a random variable describing the state of the cell, that can be either "static" or "dynamic". The state is estimated by comparing the raw laser data to the previous occupancy grid and ego-motion data. The laser data is helped by radar data, which despite being spatially less accurate has the advantage of being able to measure the speed of the detected object by exploiting the Doppler effect. The representation allows for multi-object tracking, where each object is associated with the cells it is occupying. As a final example for dynamic approaches, we can mention (Danescu, Oniga, & Nedevschi, 2011), that proposed a particle-based occupancy grid. Each cell of the grid can hold a finite number of particles, which model multiple point hypothesis of detected obstacles. The occupancy probability of a cell is defined as the ratio between the number of particles in it and the maximum number allowed. Particles have a position and velocity, and move accordingly at each timestep, with some random noise added. At every measurement, particles are weighted based on the detected occupancy of their cell. If a cell is estimated as occupied (by stereo reconstruction data), particles that are on it get a high weight and vice-versa. After the weighting, a resampling phase decides on whether to discard or multiply particles based on their weight. Particles with low weight can be discarded, while particles with high weight are multiplied. A new cell is initialized as empty if it is detected as free. Conversely, if a new cell that appears occupied, the approach creates a small set of random particles on that cell, with random velocities taken from a distribution of reasonable values. Particles that go outside the representation range get deleted. Finally, object segmentation and tracking can be performed by clustering the particles in the representation.

In order to use occupancy grids for trajectory planning, information about free space area has to be extracted from the grid. There are several approaches to achieve it. The most common is global thresholding, which has the drawback to deliver very irregular shapes for free area. For this reason, later approaches try to fit a regular curve to the free area boundary. For example, (Schreier, Willert, & Adamy, 2013) presented a Parametric Free Space Map, which models free space with a combination of parametric curves and geometric primitives. After applying a threshold on the occupancy grid, the free area undergoes a series of spatial transformations: a 2x2 median filter is applied, and subsequently morphological erosion is employed, which aims to exclude those areas that are too narrow for the ego-car to traverse. The resulting free areas are labelled, and the area in front of the ego-car is selected as the most relevant one. Finally, morphological dilation is employed to get the area back to its original size.

Researchers have also employed concepts similar to occupancy grids to encode different environmental properties, sometimes by using unusual grid formats. For example, (Sivaraman & Trivedi, 2014) proposed a Probabilistic Drivability Map, a grid-based representation where each cell holds a drivability value, that is the probability that the cell can be driven by the ego-vehicle. The cells in the grid are shaped as quadrilaterals, whose shape follows the detected lane markings on the road. The length of the cells is fixed as one car length, implying that a drivable cell should fully accomodate the ego-vehicle. The lateral boundaries of each cell follow the lane markings, whose type also influence the drivability of cells: cells that lie beyond a continuous lane marking are not drivable by the ego-vehicle, even if they are not occupied. As another example, (Weiherer, Bouzouraa, & Hofmann, 2013) employed the concept of interval maps. i.e. maps that are discretized in longitudinal direction and continuous in lateral, to create an interval occupancy map. The concept of interval map arises from the observation that many ADAS-related tasks require a much higher precision in lateral direction rather than in longitudinal (with respect to the ego-car orientation). These maps allow to encode different spatial information , such as points and areas, as well as higher properties like occupancy. By sacrificing longitudinal accuracy, this representation is significantly faster to compute with respect to standard 2D grids.

Even if roads are usually roughly approximated as a two-dimensional environments, a higher number of dimensions can be useful. In fact, in order to represent complex road environments (e.g. bridges or tunnels), two dimensional grids fall short. For these reason, researchers have investigated higher dimension representations, or adaptations of 2D representations that are able to encode height information (usually called 2.5D representations). In this direction, we can mention (Kang & Chung, 2011), which developed a Probabilistic Volume Polar Grid Map, based on stereo-vision. It is a polar grid where each cell holds a list of volumes (hexahedrons) that represent point hypothesis. The representation is analyzed to compute the free space, as well as the first obstacles on all directions. Volumes can be determined to be obstacles based on their size, location and point density. The likelihood of being an obstacle increases with size and density. The analysis divides the space into three: ground, traversable, and upper, with two transition areas between them. If a volume is more likely to be an obstacle if it is located in the traversable space.

Another interesting 2.5D representation is the Stixel World (Badino, Franke, & Pfeiffer, 2009), an image-based representation that models vertical surfaces with a set of narrow rectangles, whose height encodes the height of the object they represent. Stixels are computed by first creating a column-disparity occupancy grid, and then estimating free space by dynamic programming. The free space boundary is used as the base location for stixels. This representation is robust and very compact, and was also used in the BERTHA experiment.

As a more versatile approach, (Triebel, Pfaff, & Burgard, 2006) introduced the concept of multi-level surface map, which is a representation where every patch of a two-dimensional grid holds a list of detected surfaces, defined as their estimated height and variance. Vertical objects are represented by assigning a depth value to their surface. This representation is able to represent and compute the traversability of common complex road structures.

As for full 3D representations, (Broggi, Cattani, Patander, Sabbatelli, & Zani, 2013) used a voxel-based representation for obstacle detection, created using stereo vision. The authors point out that 2.5D representations have issues in recognizing and representing unconventional 3D structures, in particular concave surfaces. Voxels are created from a disparity-based 3D point cloud which is interpolated between current and past frames, and obstacles are segmented by color clustering. Subsequently, obstacles are tracked in order to estimate their speed.

The approach we present in the next chapters focuses on detecting road as a semantic entity, rather than free space. As such, our approach differs from occupancy grids as they approximate road with *drivable area*, while we do not. However, similarly to many occupancy grids, we use a 2D Cartesian grid, where a probability is computed independently on each cell. In our case, that probability estimates whether the cell is part of the road or not, rather than whether it is occupied or not.

2.3 Trajectory Planning

Most state-of-the-art autonomous driving systems need a detailed environment representation to plan their trajectories on. For example, the participants of the 2011 DARPA Urban Challenge were given road maps of the full site by the organisers (Bacha et al., 2008)(Miller et al., 2008)(Urmson et al., 2008). The trajectories planned by state-of-the-art systems have a time horizon in the order of a few seconds, depending on the size of the available representation, and on the speed of their computation algorithm (the longer the time horizon, the slower is the computation). When the system has to drive from a point A to a point B far away, the task is divided into smaller sections, based on checkpoints that are usually taken from a topological map. The system plans short term maneuvers between each checkpoint once it reaches them. The way they perform this task can vary significantly between different systems, but there are two common approaches: computing the trajectory as an analytical function, or computing it as a collection of nodes, by a tree search algorithm. Both approaches have to employ a cost function to assign a cost to each location of the planning space (which, for advanced algorithms, can have many additional dimensions), and this function depends on the environment representation available. An important point is that the resulting trajectory has to be drivable by the ego-vehicle in terms of actual vehicle dynamics. If the trajectory is computed analytically, dynamics can be incorporated as constraints in the process and the resulting trajectory is slower to compute (and update to sudden environment changes) but smooth and drivable. If the trajectory is a collection of nodes, it is quicker to compute and update, but requires a second step to make the trajectory drivable by a real vehicle. The approach chosen often depends on the type of representation the system employs: analytical road representations (e.g. road boundaries estimated by b-spline) call for an analytical solution, while for grid-based representations tree-search is the most natural approach. For a comprehensive review of planning approaches, we mention (González, Pérez, Milanés, & Nashashibi, 2016).

An example of analytically computed trajectory is Daimler's BERTHA, which calculated them as the solution of a cost function minimization problem, where constraints were imposed by road boundaries and detected obstacles(Ziegler, Bender, Dang, & Stiller, 2014). A driving corridor was determined based on those, and the trajectory had to minimize five different costs: it had to stay close to the middle of the corridor, it had to keep the speed close to the desired one, and it had to minimize accelerations, jerk and yaw rate.

The approach presented in (Guo, Kidono, & Ogawa, 2016) is particularly interesting, because it exploits the behavior of the leading vehicle as an additional cue for the trajectory computation. The leading vehicle here is defined as the preceding vehicle with the trajectory most similar to the ego-car. Thus, it does not have to be the vehicle immediately in front of it. Each detected vehicle gets a score, which represents the affinity between its trajectory and each lane of the road. Each vehicle is assigned to the lane with the highest score. The score comprises three terms: the overall distance between trajectory and lane, the similarity between trajectory and lane, and the agreement between the vehicle future movement (for a short time interval, assuming it will maintain its current speed and direction) and lane. Subsequently, a second score is assigned to the vehicle which have been associated to the ego-vehicle lane. This second score is computed similarly to the first, but measures the affinity with the trajectory of the ego-vehicle itself. The vehicle with the highest score is determined as the leading vehicle. In this work, the ego-car trajectory is computed as a clothoid, where every control point is represented as a point mass, connected with the others with springs and dampers. On these masses act three external forces: an attractive force pointing to the center of the host lane, another attractive force pointing to the leading vehicle trajectory, and finally a repulsive force avoiding surrounding vehicles. This solution helps creating a more human-like trajectory, especially during maneuvers to avoid obstacles, where the leading vehicle provides a good evasion maneuver proposal.

The approaches that compute a trajectory by employing a tree of spatial nodes need to use a tree search algorithm to find the optimal path. The most common of these algorithms is by far A^*. A^* (Hart, Nilsson, & Raphael, 1972) is a best-first search algorithm, i.e. it explores those nodes that *seem* to be the most promising ones. To determine how promising a node is, A^* needs to establish a heuristic that expresses a prior cost to a node n. This heuristic, usually indicated as $h(n)$, is admissable only if it never overestimates the true cost of any node. The heuristic used depends on the actual problem considered. The search prioritizes nodes with the lowest value $f(n)$, where

$$f(n) = g(n) + h(n) \tag{2.1}$$

$g(n)$ is the actual cost of the node, including all the past nodes that led to it. The cost is calculated depending on all the various environment properties of the spatial area included in the node (e.g. road, lanes, obstacles, etc.), and the formula to compute it is a key aspect of any approach.

Due to its performances, A^* is the basis of many different variants. In fact, the vast majority of vehicles competing in the DARPA Urban Challenge 2007 relied on A^* or on its variants for trajectory planning in unstructured environments (Dolgov, Thrun, Montemerlo, & Diebel, 2010). For example, *Boss* (Ferguson, Howard, & Likhachev, 2008) relied on a variant of A^* called Anytime Dynamic A^* or Anytime D^* (Likhachev, Ferguson, Gordon, Stentz, & Thrun, 2005), which is particularly apt to deal with sudden changes in environment representation (e.g. new obstacles detected). Every time a significant change is detected, the algorithm does not recompute the solution from scratch, but tries to repair the previous solution by looking for deviations in the local area of the change itself. This approach ensure a very quick reaction to obstacles, and a trajectory that does not change dramatically over different iterations. They propose two heuristics: one that computes the distance to the goal by taking only the ego-vehicle motion dynamics into account but not the environment features (i.e. the environment is considered empty and all drivable, and this heuristic can be computed in advance offline for any location and stored in a lookup table), and another heuristic that depends on the shortest path that traverse drivable space, without taking into account car dynamics (considering the actual drivable area in the representation). The algorithm uses the maximum value between these two heuristics.

Junior (Montemerlo et al., 2008) used a variant of A^*, which they call Hybrid A^*. This algorithm has the same underlining concept of A^*, but differs in the way it generates nodes. In this approach, nodes can be generated at any location of a continuous search space (whereas classical A^* only uses the center of cells), based on the dynamics of the ego-vehicle. Paths from node to node can be curved according to the motion constraints of the ego-vehicle. This approach guarantees that the computed path is drivable (by construction), but since the reachable state space becomes infinite (as the location of each node can vary freely), it cannot guarantee search completeness and minimal-cost solution. This approach uses two heuristics that are equivalent to the ones used by *Boss* (but independently developed), which they call ”non-holonomic-without-obstacles” and ”holonomic-with-obstacles”.

For our own trajectory planner, we employ another variant of A^*, called Fringe search (Björnsson, Enzenberger, Holte, & Schaeffer, 2005): This algorithm uses the same concept used by $A*$ of estimating the remaining cost to reach the target from each node in order to select which nodes to expand first, but unlike $A*$ it iterates over two lists, *now* and *later*, which store respectively the current and next iteration. The planner goes through the nodes stored in *now*, expands them and inserts their children in one of the two lists depending on their expected cost. If it is lower than a threshold, they get inserted in *now*, while if it is higher they go into *later*. Once the iteration over *now* ends because all the remaining nodes have an estimated cost higher than a threshold, the algorithm increases the threshold, moves all the nodes from *later* to *now*, and moves to the next iteration. The main advantage of Fringe Search is that these lists do not need to be sorted, unlike what happens for A^*, which needs to sort all open nodes by cost, a process that can be very time-consuming.

Chapter 3

Road Probability Framework

Chapter overview *In this chapter we present our probabilistic framework, that fuses direct and indirect detection to infer road probability in a 2D grid map.*

3.1 Motivation

The core concept of our approach is to combine a direct road detection method with an indirect method that takes advantage of one of the main problems of direct visual road detection systems: other vehicles in the scene, that create occlusions. Our reasoning is that the motion of those vehicles holds valuable information that can potentially cover the loss of direct visibility. By observing and interpreting their motion, we can infer information from their representations to complement our own one. The basic assumption of our approach is that all vehicles move on road area. Therefore, all patches that are traversed by a vehicle must be road. Similarly, if we can estimate which patches will be traversed by the vehicle in the future, or have been in the past, we can infer the road probability of those patches as well. In this chapter we build a probabilistic framework that uses information from direct and indirect detection to infer a road probability for each location of a 2D grid map. We will call each location a "patch". All the computations are carried out patch-wise, ignoring any spatial interaction between different patches. Every patch is computed independently from its neighbors, but that does not mean they bear no relation between each other. In fact, the inputs they receive do carry a spatial relation, so that ultimately the road probability of neighboring patches is likely to be similar, even if this spatial relation is not explicit in the probability formulation.

3.2 Road Probability Formula

In this section we derive the main road probability formula that is used independently on every patch. We denote patches using only one index i, instead of using two, in order to improve readability. The road probability has to be computed from our inputs, which

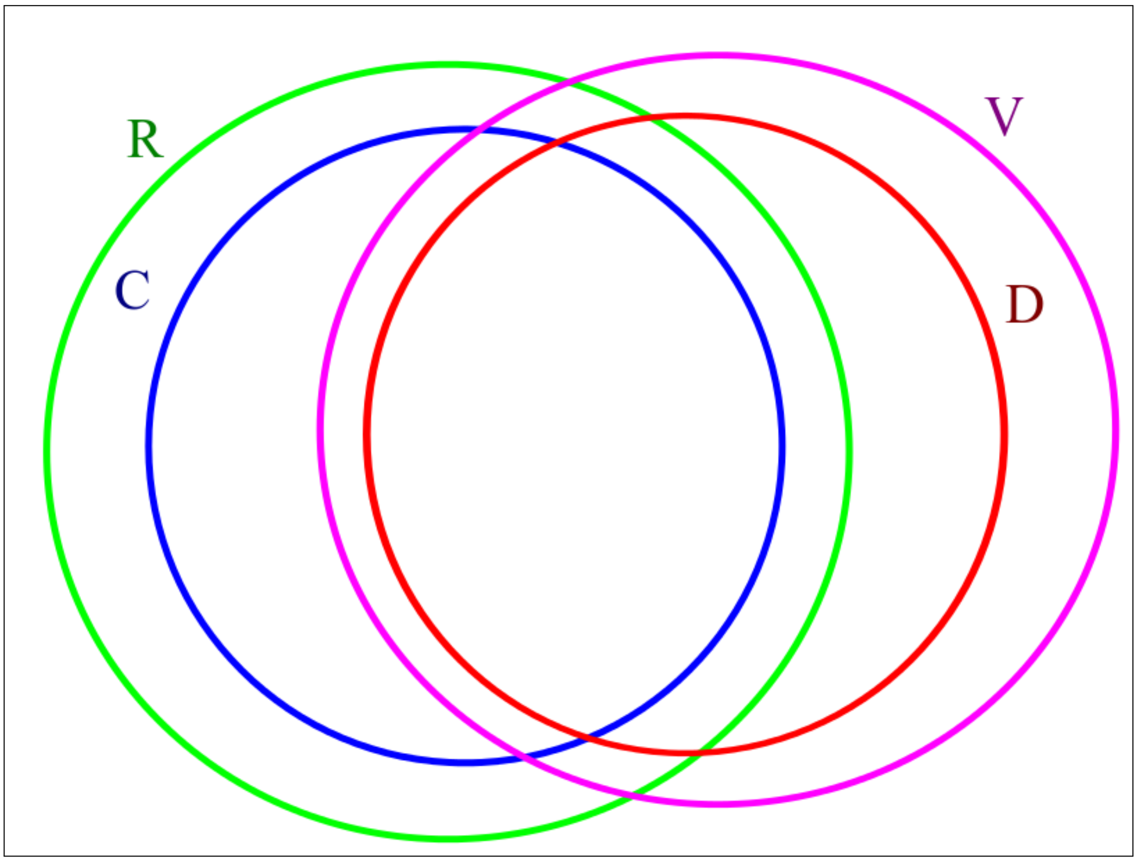


Figure 3.1: Euler diagram of the four events.

are direct-detection-based road confidence and position and velocity of detected moving vehicles in the scene.

We define four events:

- **Event R** The event that the considered patch is part of the road. In the notation we use, $p(R_i)$ is the probability that patch i is road, and our goal is to compute it as a function of our inputs.

- **Event C** The event that the considered patch has been, is or will be traversed by at least one moving road vehicle. The time window in which C is considered is not necessarily relevant for our approach, but obviously $p(C_i)$ can be realistically estimated only in a reasonably limited time window. This estimation is a key aspect of the approach, and will be discussed in detail in the next chapter. For simplicity, we assume that if a vehicle traverses a patch, then that patch is part of the road. In this case, we can state that $C \subseteq R$, and thus $p(R_i|C_i) = 1, \forall i$.

- **Event V** The event that the considered patch is visible to the direct detection system. We assume that V is independent from R, so that $p(R_i \cap V_i) = p(R_i)p(V_i)$.

- **Event D** The event that the considered patch has been correctly evaluated by the direct detection system. Since this can happen only if the patch is visible, we can consider $D \subseteq V$, and thus $p(V_i|D_i) = 1, \forall i$. Conversely, $p(D_i|V_i)$ expresses the reliability of the direct detection system. $p(R_i|D_i)$ is the road confidence value provided by said system, and it is one of the inputs of our approach. For simplicity, we assume that D is independent from C, so that $p(D_i \cap C_i) = p(D_i)p(C_i)$. It could be argued that there is a possible dependency between the two events, since direct detection always fails for patches that are directly under a vehicle. However, this dependency only influences a very narrow subset of C and it is unclear how to quantify it. Additionally, this effect is already covered by event V: patches directly under a vehicle are obviously not visible, so D will be always false on them as a consequence. This is why we can simplify the relation between C and D as independent events.

Figure 3.1 shows an Euler diagram of these events. Our objective in this section is to obtain a formulation of $p(R_i)$ in terms of our inputs. In order to do so, we apply the law of total probability by splitting R into $R \cap \overline{D}$ and $R \cap D$:

$$\begin{aligned} p(R_i) &= p(R_i \cap D_i) + p(R_i \cap \overline{D}_i) \\ &= p(R_i|D_i)p(D_i) + p(R_i|\overline{D}_i)p(\overline{D}_i) \\ &= p(R_i|D_i)p(D_i) + p(R_i|\overline{D}_i)\left[1 - p(D_i)\right] \end{aligned} \tag{3.1}$$

Focusing now on the term $p(R_i|\overline{D}_i)$, we want to express it with respect to event C.

$$\begin{aligned} p(R_i|\overline{D}_i) &= \frac{p(R_i \cap \overline{D}_i)}{p(\overline{D}_i)} \\ p(R_i \cap \overline{D}_i) &= p(R_i \cap C_i \cap \overline{D}_i) + p(R_i \cap \overline{C}_i \cap \overline{D}_i) \\ &= p((R_i \cap C_i) \cap \overline{D}_i) + p(R_i \cap \left(\overline{C}_i \cap \overline{D}_i\right)) \end{aligned} \tag{3.2}$$

Assuming C and D are independent, and recalling that $R_i \cap C_i = C_i, \forall i$ by construction, we have:

$$\begin{aligned} p(R_i \cap \overline{D}_i) &= p(C_i \cap \overline{D}_i) + p(R_i| \left(\overline{C}_i \cap \overline{D}_i\right))p(\overline{C}_i \cap \overline{D}_i) \\ &= p(C_i)p(\overline{D}_i) + p(R_i| \left(\overline{C}_i \cap \overline{D}_i\right))p(\overline{C}_i)p(\overline{D}_i) \\ &= \left[p(C_i) + p(R_i| \left(\overline{C}_i \cap \overline{D}_i\right))p(\overline{C}_i)\right] p(\overline{D}_i) \end{aligned} \tag{3.3}$$

To which immediately follows

$$\begin{aligned} p(R_i|\overline{D}_i) &= \frac{\left[p(C_i) + p(R_i| \left(\overline{C}_i \cap \overline{D}_i\right))p(\overline{C}_i)\right] p(\overline{D}_i)}{p(\overline{D}_i)} \\ &= p(C_i) + p(R_i| \left(\overline{C}_i \cap \overline{D}_i\right))p(\overline{C}_i) \end{aligned} \tag{3.4}$$

The term $p(R_i|\left(\overline{C}_i \cap \overline{D}_i\right))$ will be discussed in section 3.4. Substituting (3.4) into equation (3.1), and recalling that $p(D_i) = p(D_i|V_i)p(V_i)$, leads to our main road probability formula:

$$p(R_i) = p(R_i|D_i)p(D_i|V_i)p(V_i) + \left[p(C_i) + p(R_i|\left(\overline{C}_i \cap \overline{D}_i\right))p(\overline{C}_i)\right]\left[1 - p(D_i|V_i)p(V_i)\right] \tag{3.5}$$

This equation features two main terms, modulated by $p(D_i|V_i)p(V_i)$ and its negation. The first term, $p(R_i|D_i)$, is the direct detection contribution. The second, $p(C_i) + p(R_i|\left(\overline{C}_i \cap \overline{D}_i\right))p(\overline{C}_i)$, is the indirect detection contribution (see sections 3.3 and 3.4). The modulation via $p(D_i|V_i)p(V_i) = p(D_i)$ represents a trade-off between direct and indirect detection, depending on the estimated probability that direct detection is correct, and how much it is to be trusted (see section 3.5).

3.3 Behavior Interpretation

The indirect detection contribution (that we call "Behavior Interpretation") in our framework is based on the event C_i. This is the event that patch i is traversed by a moving vehicle. The event is not dependent on time: it can happen in the future, present or past. Only one vehicle is necessary for the event, and in case of multiple vehicles j the event C_i is the union of the events that any vehicle j traverses patch i, $C_{i,j}$. Thus, we can write:

$$C_i = \bigcup_j C_{i,j} \tag{3.6}$$

For simplicity, we assume that $C_{i,j}$ are all independent from each other. This is a strong assumption, since it is clear that there can be situations where the trajectories of two vehicles depend on each other (they want to avoid collisions), but since our event C ignores the time component of trajectories, this interaction is much less significant. For example, consider figure 3.2. Here, vehicles A, B and C are traversing an intersection. Vehicle A is heading straight ahead, while vehicles B and C are on the opposing lane and want to turn to their left, intersecting with the path of vehicle A. As these three vehicles are on a collision course, we can expect the drivers to change their trajectories to avoid a crash. In this sense, the trajectories of the vehicles are obviously not independent from each other. However, the events $C_{i,j}$ associated to them are inter-dependent only in time domain, and not in space. Vehicle B does not modify the space component of its trajectory, but it simply brakes instead, letting vehicle A pass and only then eventually resuming its motion. The patches that vehicle B traverses are not changed by vehicle A, only the timing is. The same reasoning can be applied to vehicle C: it brakes in order not to hit vehicle B, but it does not change the spatial component of its trajectory.

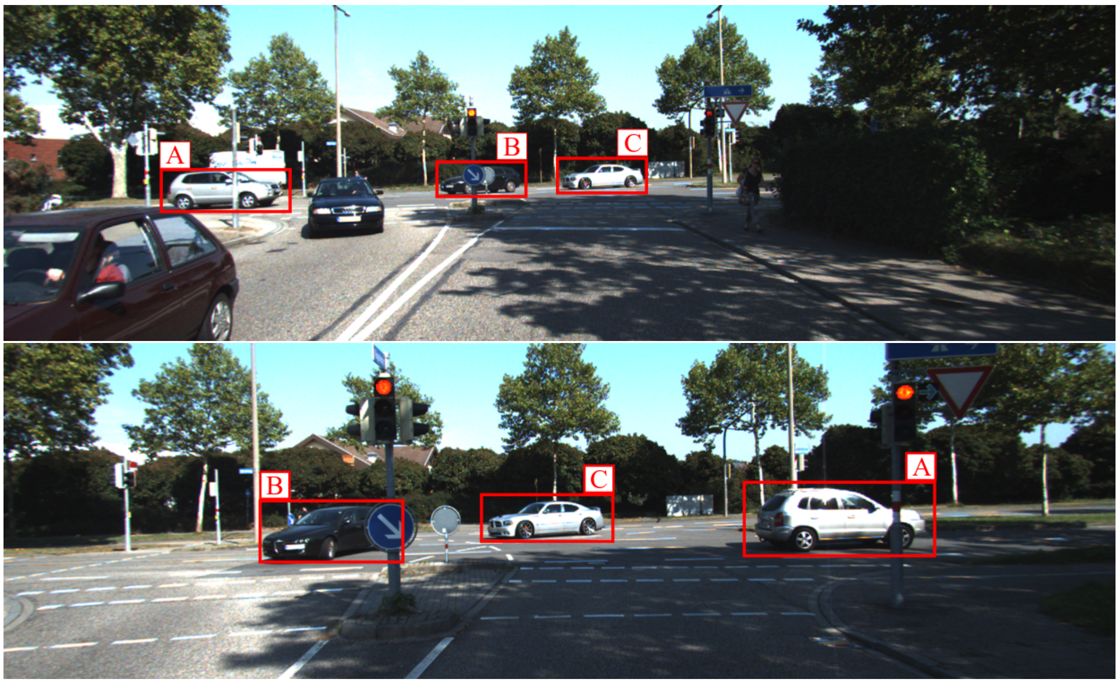


Figure 3.2: Intersection scene, where the trajectories of vehicles A, B and C are not independent. However, the inter-dependency is solely in the time domain.

Hence, while trajectories can definitely change depending on other trajectories, in many instances only their time component is affected, and since the definition of event C does not include any constraint about time, we can ignore this dependency and treat $C_{i,j}$ as independent events. Since we treat them as independent events, their probabilities sum in the following fashion:

$$\begin{aligned} p_n(C_i) &= p(C_{i,1}) + p(C_{i,2}) - p(C_{i,1})p(C_{i,2}) && \text{if } n = 2 \\ p_n(C_i) &= p_{n-1}(C_i) + p(C_{i,n}) - p_{n-1}(C_i)p(C_{i,n}) && \text{if } n > 2 \end{aligned} \tag{3.7}$$

3.3.1 Trajectory Cloud

The probability $p(C_{i,j})$ can be estimated with various methods. However, most state-of-the-art methods aim to predict the trajectory of a vehicle, whose concept is different from the patch-based event C, so they would need to be modified to fit with our time-free definition of said event. Alternatively, if a sufficiently rich database is available, it would be possible to build an empirical function by observing the motion of a large number of actual vehicles in different scenarios. In this work we chose to employ a simple kinematic model, using a 2D function, that provides a probability value given the relative location

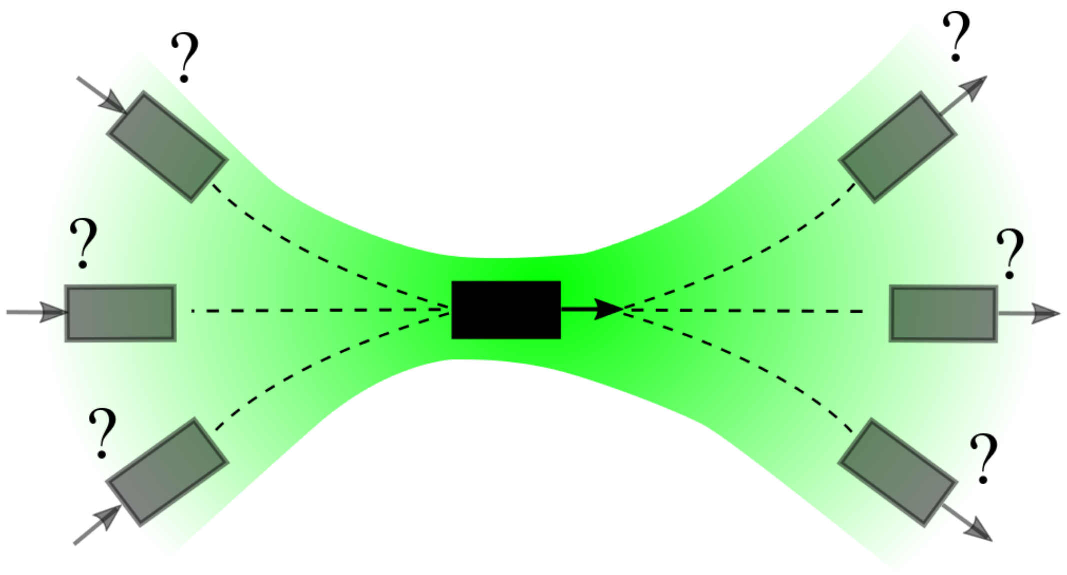


Figure 3.3: Concept of trajectory cloud. The detected vehicle has many possible trajectories, that cover a certain area with different probability (represented as shades of green).

of the patch i in the reference system of vehicle j (x is the pitch axis and z is the roll axis), and the speed of the vehicle itself. Nevertheless, it is important to keep in mind that the validity of the whole framework does not depend on the particular model used for $p(C_{i,j})$.

Vehicle j can have many possible trajectories, as its yaw and speed can vary freely in time. Our probability function has to model all possible trajectories, without the time constraint. In fact, we only care about whether vehicle j can traverse patch i, and not when. The resulting function should look like a trajectory "cloud", that comprises all possible trajectories (figure 3.3). Since we want to estimate both past and future positions of the vehicle, we use a function symmetrical with respect to the x axis of the vehicle's reference system. Intuitively, the function produces high values immediately in front of the vehicle, because those patches will be traversed almost certainly. The values fade away with distance, as the vehicle could change its direction and thus potentially cover a wider area. We chose to keep this probabilistic approach also under the car, so that we can model possible position errors due to inaccurate detection. Every vehicle has a rectangular shape, with a certain direction and speed. If the detection was perfectly accurate our p(C) function should just have the shape of the vehicle itself, with a 100% probability on the patches covered. However, every kind of detection yields some form of error. To account for that, we model the vehicle in a simplified way as a set of single points along its x-axis. Due to detection error, we can assume every point has a Gaussian-like probability of being on the location it was detected on. The standard deviation of this probability is one of the parameters of our system. Since the probability function has the same shape for all points, we integrate it along the x-axis, between the boundaries of the rectangle. The resulting shape is the one of a difference of two error functions. Regarding the z-axis, the probability is constant within the rectangle boundaries. Beyond them, we need to take into account the possible maneuvers of the vehicle: it could be heading straight, or turn left or right. Our solution is to keep the function obtained by integration, and just spread it along the x-axis by progressively increasing the variance. This way decreases the further away we look at. The boundaries spread is modeled as dependent on the vehicle speed. The higher the speed, the slower the variance increase, since the vehicle will be less likely to turn at high speed. The function is formulated as follows:

$$p(C_{i,j}) = \frac{1}{2}\left[\operatorname{erf}\left(\frac{x_{i,j}+\eta_j}{\sqrt{2}\sigma_{i,j}}\right) - \operatorname{erf}\left(\frac{x_{i,j}-\eta_j}{\sqrt{2}\sigma_{i,j}}\right)\right] \tag{3.8}$$

$$\sigma_{i,j} = \begin{cases} \omega_j & for\, |z_{i,j}| \le \frac{L_j}{2} \\ \omega_j + A\frac{\left(z_{i,j}-\frac{L_j}{2}\right)^2}{|\mathbf{v}_j|^2} & for\, |z_{i,j}| > \frac{L_j}{2} \end{cases} \tag{3.9}$$

Here $[x_{i,j}, z_{i,j}]$ is the position of patch i in the reference system of vehicle j, L_j is the length of the vehicle j, η_j is a parameter related to the width of it, ω_j is related to the uncertainty of its position, and A is a tuning factor. A contour plot of this function with two different values of A is shown in figure 3.4. A controls the rate at which the variance increases, and thus letting the function be as wide as desired at long distances.

The function estimates the patches that should be traversed by the vehicle in the future (in front of it), in the present (under it), and in the past (behind it). However, the entirety of the function domain is not used at every iteration. In fact, the "past" estimation is employed only as soon as a vehicle appears (and an estimation of its speed is available). This estimation will not change further, as any future will not tell us more about the vehicle's past. Conversely, present and future estimations are computed at every iteration while the vehicle is visible. In fact, every new observation provides more accurate information. This reasoning leads to a different management of the three different domains that will be explained in the following subsection.

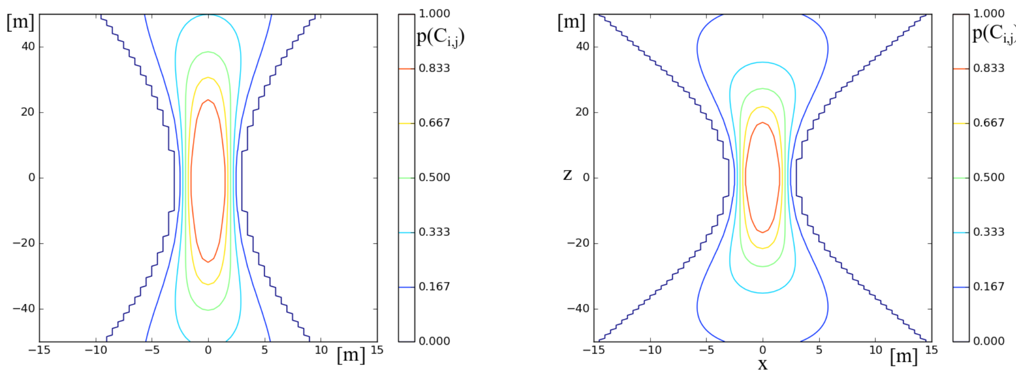


Figure 3.4: Contour plot of $p(C_{i,j})$, with j being a vehicle centered in $[0, 0]$ and facing upwards, using two different parameters A. On the left $A = 0.01$, while on the right $A = 0.02$.

3.3.2 Temporary and Permanent Estimations

As the detected position and velocity of vehicles change at every measurement, it is necessary to define a way to update the estimated $p(C_{i,j})$. For this purpose we distinguish two main cases: temporary and permanent estimations. Temporary estimations $p(C_{i,j})^{\sim}$ need to be refreshed at every iteration, as they depend on the current state of the observed vehicles. The estimated future positions of vehicles are considered temporary by our system. At every time step, the previous estimation will be dropped and replaced with a newer one. Conversely, permanent estimations $p(C_{i,j})^{\infty}$ are those that depend on the state of the vehicle in a particular instant, and thus do not change with further measurements. Estimated past and present positions are permanent: they are computed only once, and will never change, no matter the successive behavior of their vehicle. To handle this difference, each patch of the representation grid does not only hold a probability value, but also two lists: the first list includes all the vehicles that have a temporary estimated probability to traverse the patch in the future, along with the value $p(C_{i,j})^{\sim}$ itself. The second list includes all vehicles that have a permanent probability to have traversed the patch, again with $p(C_{i,j})^{\infty}$. When a vehicle disappears from the scene, its future $p(C_{i,j})^{\sim}$ does not disappear with it, but instead is just moved to the permanent list and changed into $p(C_{i,j})^{\infty}$. The disappearance of a vehicle can be a very useful event: it means that it went to a road we cannot see, and therefore we want to save the last estimation we had of the area it was heading to. It will not change anymore, if we assume the vehicle has disappeared for good. If it does reappear (i.e. the disappearance was due to a misdetection), the system will be able to retrieve its $p(C_{i,j})$ and prevent inconsistencies. After some time, each permanent $p(C_{i,j})^{\infty}$ are merged into a single value $p(C_i)^{\infty}$, in order to avoid an unreasonable memory burden for the system.

3.4 Unpredicted Maneuvers

The term $p(R_i|\overline{C}_i \cap \overline{D}_i)$ in (3.5) is a very important element of the road probability formula. It is the probability that patch i is road, in the case that none of the vehicles in the scene traverses it and the direct detection does not provide reliable information. This term could be interpreted simply as the prior probability P_0 of a patch being road, as other vehicles and direct detection do not provide information about it. It could be set to 0.5 if we do not know anything, or it could be set to the average road/not-road ratio in the environment we are in, if we have data about it. However, we can realize that the term $p(R_i|\overline{C}_i \cap \overline{D}_i)$ does not have to be the same value for every patch i. In fact, there are many cases where it is evident it really should not. For example, consider figure 3.5, which shows a painted roundabout partially occluded by a vehicle. The incoming vehicles initially point to the center of the roundabout, so that $p(C_i)$ will be high there, and so will $p(R_i)$. However we will soon observe them entering the roundabout and changing direction accordingly. All of them will avoid the center of the roundabout itself, because it is not road. This is a behavior that provides information not only about the road area that was actually traversed by the vehicles, but also about the center of the roundabout. All those vehicles were supposed to traverse the center of the roundabout, but instead they all actively avoided it, giving us strong evidence that said area is not road. We can think that patches in the center of the roundabout had a high probability to be traversed by a vehicle at some point, and this fact should lead to a low probability for those patches to be road in case they end up not being traversed at all. Conversely, patches that were not predicted to be traversed (e.g. patches outside the roundabout) are not affected by this behavior. This reasoning is useful also in other situations, and is used by human drivers too. If we observe multiple vehicles that make unexpected maneuvers to avoid a certain area we cannot see well, we instinctively prepare to avoid that area ourselves. There could be a pothole, or something we do not want to run over. At the very least, we will slow down and wait to be able to see that area well before traversing it.

As a less obvious example, consider figure 3.6. It shows a vehicle appearing from a road on the left side, apparently heading straight towards an area we cannot see on the right. However, the vehicle steers away before entering that area. In this situation, the fact that the vehicle did not traverse, for example, the courtyard at our left does not give us any information about it, because the trajectory of the vehicle was always far from that area. However, the fact that it did not traverse the area it was initially predicted to drive on is a hint we can consider. The vehicle was headed there, but it avoided it. This behavior can suggest that said area is not road, although the information provided by a single vehicle is not sufficient for us to be sure of it. This is an important difference between inferring where road is and inferring where it is not: while we just need one vehicle traversing a patch to state that the patch is road, the same cannot be said about the opposite. The more vehicles avoid the same patch, the more information about it

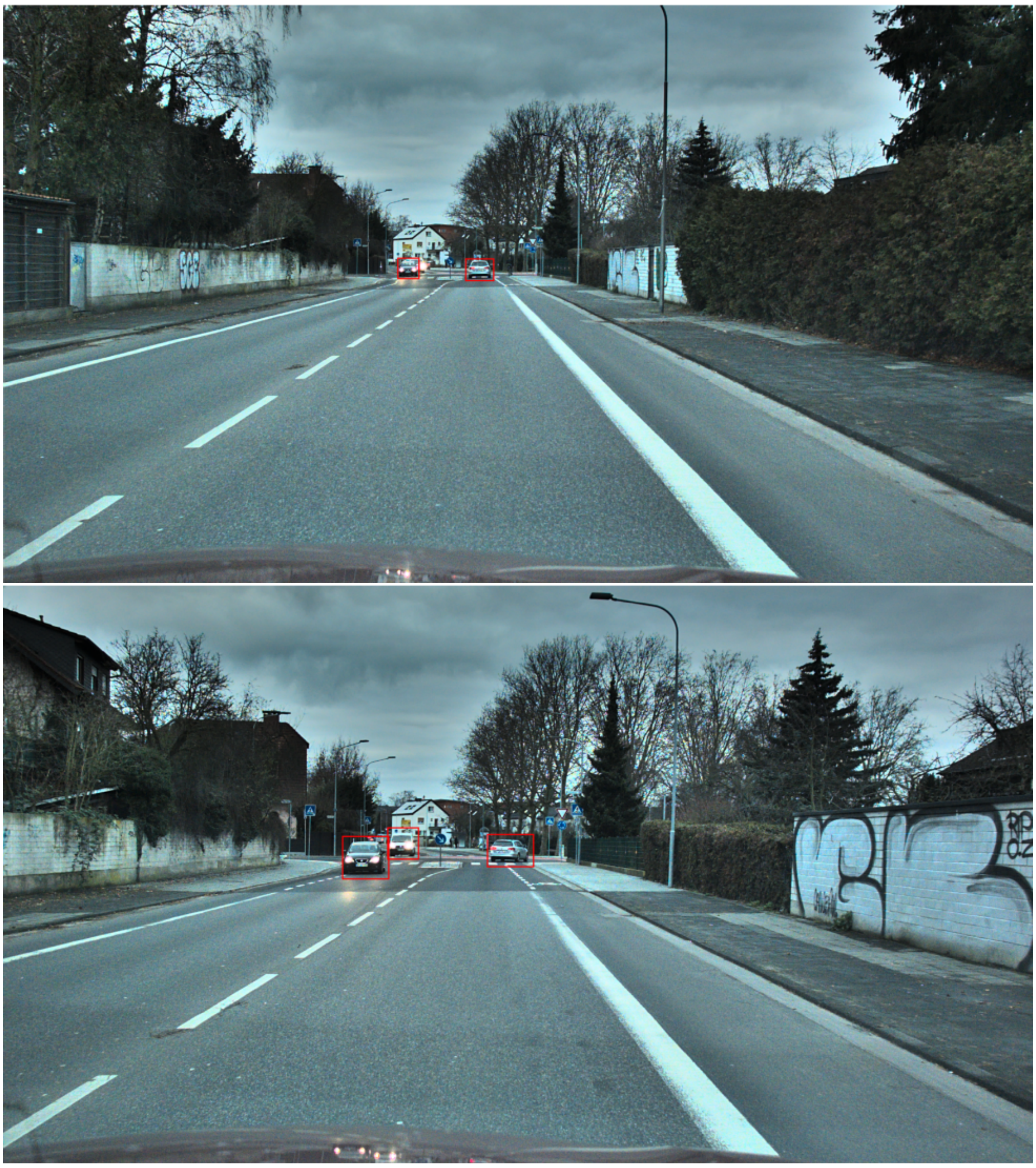


Figure 3.5: A small urban roundabout, where the unpredicted maneuvers of the vehicles marked in red, avoiding the center area, give us evidence that the center is not road, even if we cannot see it properly.

they give us.

In general terms, $p(R_i|\overline{C}_i \cap \overline{D}_i)$ (which we call "Avoidance") should be low for patches that are predicted to be traversed, while it should be equal to a prior probability P_0 if there are no vehicles that seem to be heading there. In order to model this reasoning,

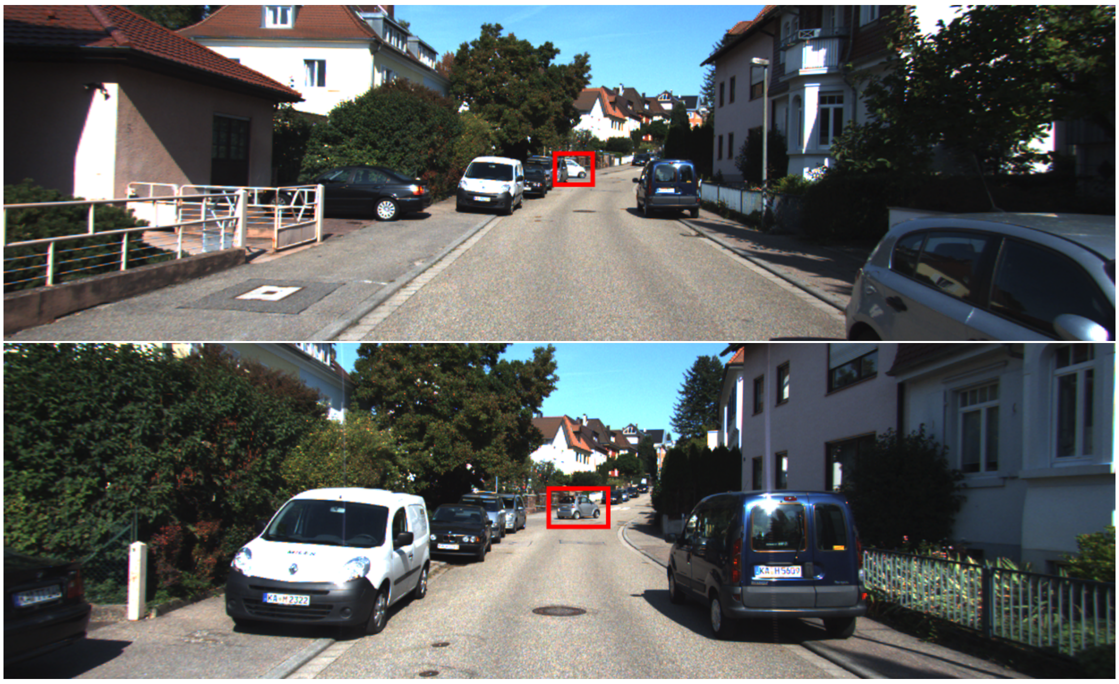
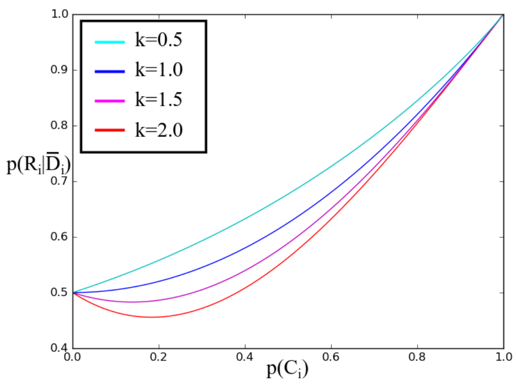


Figure 3.6: Urban intersection, where the maneuver of the vehicle marked in red decreases the probability of the presence of a side road we cannot see.

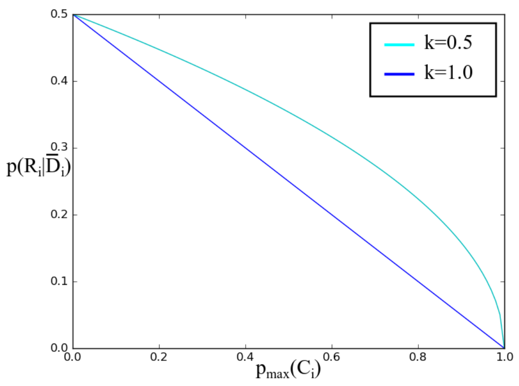
we want the avoidance to be dependent on $p(C_i)$. However, the importance of this term is highest when vehicles avoid a patch, i.e. when their trajectory cloud changes. For this reason, we want the term to be dependent only on the part of the trajectory cloud that can actually change, i.e. $p(C_i)^\sim$ (see section 3.3.2). If we make the avoidance be dependent on the current value of $p(C_i)^\sim$, the effect will be lost as soon as the vehicles perform their maneuvers and our system does not predict them to traverse the patch anymore. In order to keep this effect after the maneuver, our solution is to have the avoidance being dependent on the maximum value of $p(C_i)^\sim$ over time, i.e. $p_{max}(C_i)^\sim$. This ensures that the effect stays after maneuvers are completed. We model the avoidance, $p(R_i|\overline{C}_i \cap \overline{D}_i)$, with the following function:

$$p(R_i|\overline{C}_i \cap \overline{D}_i) = P_0\left(p_{min}(\overline{C}_i)^\sim\right)^k = P_0\left(1 - p_{max}(C_i)^\sim\right)^k \tag{3.10}$$

Where P_0 is the prior probability and $k > 0$ a tuning factor. If there are no cars in the scene, or if no cars are predicted to traverse patch i, the term equals to the prior probability, while it decreases the likelier a patch is to be traversed. Figure 3.7a shows the effect of the k parameter on the values of the overall behavior interpretation term $p(R_i|\overline{D}_i) = p(C_i) + p(R_i|\overline{C}_i \cap \overline{D}_i)p(\overline{C}_i)$ as a function of $p(C_i)$, with $p_{max}(C_i) = p(C_i)$, which is the case where the avoidance maneuver has not happened yet. The graph shows that for $k > 1$ the term has a minimum with $p(C_i) \neq 0$, which means that at certain



(a) $p(R_i|\overline{D}_i)$ for no unpredicted maneuvers, with $p(C_i) = p_{max}(C_i)$



(b) $p(R_i|\overline{D}_i)$ after unpredicted maneuvers, with $p(C_i) = 0$

Figure 3.7: (a): Value of the behavior interpretation term with different values of k in a scenario without (or before) unpredicted maneuvers, with $P_0 = 0.5$. (b): Value of the behavior interpretation term with different values of k after unpredicted maneuvers. Note that if $p(C_i) = 0$, then $p(R_i|\overline{D}_i) = p(R_i|\overline{C}_i \cap \overline{D}_i)$

values an increase in $p(C_i)$ would lead to a decrease in the overall probability. Since we want to avoid this effect, we set $0 < k \leq 1$. Figure 3.7b shows the value of the behavior interpretation term as a function of $p_{max}(C_i)$ when $p(C_i) = 0$, which is the case when patch i was supposed to be traversed with a certain probability, but it has been avoided instead. Here $p(R_i|\overline{D}_i) = p(R_i|\overline{C}_i \cap \overline{D}_i)$, so we can see how the avoidance value decreases the higher the probability $p_{max}(C_i)$ is, modeling how the patch is unlikely to be road the more it was likely to be traversed in the first place.

3.5 Direct Detection

In order to complete the framework, we need a direct visual detection system. Any kind of direct detection system can be incorporated, as long as it provides a Birds' Eye View (BEV) confidence map that can be translated into a probability of each patch being road.

3.5.1 Visibility

In our road probability formula, $p(V_i)$ represents the chance that patch i is visible to the direct detection system. The direct detection system has to provide road confidence values in a predefined field of view, whose shape and position relative to the ego-car depends on the specifics of the chosen system, and by our definition every patch outside this area has $p(V_i) = 0$. Inside the area, the system checks for occlusions, caused by other vehicles or static obstacles detected by the sensors. If a patch lies behind an occlusion, its $p(V_i)$ is set to 0, while if it is not occluded we set $p(V_i) = 1$. The road probability formula uses the maximum value recorded, so that if a patch was visible at some point, the information is not lost when the patch gets occluded. If we denote the current visibility computed at time $t = t_0$ as $p(V_i)[t_0]$, then the visibility value $p(V_i)|_{t=t_0}$ used in the road probability formula for that iteration is

$$p(V_i)|_{t=t_0} = \max\left[p(V_i)[t]\right] \text{ for } t \leq t_0 \tag{3.11}$$

3.5.2 Reliability

The reliability of direct detection, $p(D_i|V_i)$, is computed for every patch that is considered visible by the system. The term expresses the probability that the patch can be classified correctly by the direct detection. Since we define D_i as a subset of V_i, $p(D_i|V_i)$ is not significant when the patch is not visible, as $p(V_i) = 0$. We can expect this term to vary within the FOV, as the reliability of any direct detection system will not be the same everywhere. In fact, as explained in section 1.2, direct road detection performances are very susceptible to various factors, such as road texture, illumination, etc. Furthermore, it is to be expected that the reliability of any direct detection algorithm decreases with distance (Kühnl, Tobias, Kummert, & Fritsch, 2011), as well as with viewing angle, e.g. due to the fact that pixels progressively comprise a larger area. For this reason, we model $p(D_i|V_i)$ with a spatial function dependent on the relative position of each patch in a reference frame centered on the ego-car. The function parameters depend on the expected performances of the chosen system, and can be estimated by collecting statistics on real data, or by modeling. We can expect that for most systems the function will have higher values on the center of the field of view and in the vicinity of the ego-vehicle, while it will have lower values on the sides and on the far edge of the FOV. See section 5.5 for details about the actual function we use in our experiments.

The road probability formula at each iteration uses the average $p(D_i|V_i)$ over time, using $p(V_i)$ as a weight, so that only the iterations when the patch was visible are significant. The direct detection term used in the formula, $p(R_i|D_i)$, is also the average over time of the values $p(R_i|D_i)[t]$ provided at each iteration t, using the respective values of $p(D_i|V_i)[t]p(V_i)[t]$ as weights. Therefore, we can write:

$$p(D_i|V_i)|_{t=t_0} = \frac{\sum_{t=0}^{t_0} p(V_i)[t]p(D_i|V_i)[t]}{\sum_{t=0}^{t_0} p(V_i)[t]} \tag{3.12}$$

$$p(R_i|D_i)|_{t=t_0} = \frac{\sum_{t=0}^{t_0} p(V_i)[t]p(D_i|V_i)[t]p(R_i|D_i)[t]}{\sum_{t=0}^{t_0} p(V_i)[t]p(D_i|V_i)[t]} \tag{3.13}$$

Figure 3.8 shows an example of the direct detection contribution weight, i.e. $p(D_i) = p(D_i|V_i)p(V_i)$, showing that the area immediately in front of the sensor is typically not

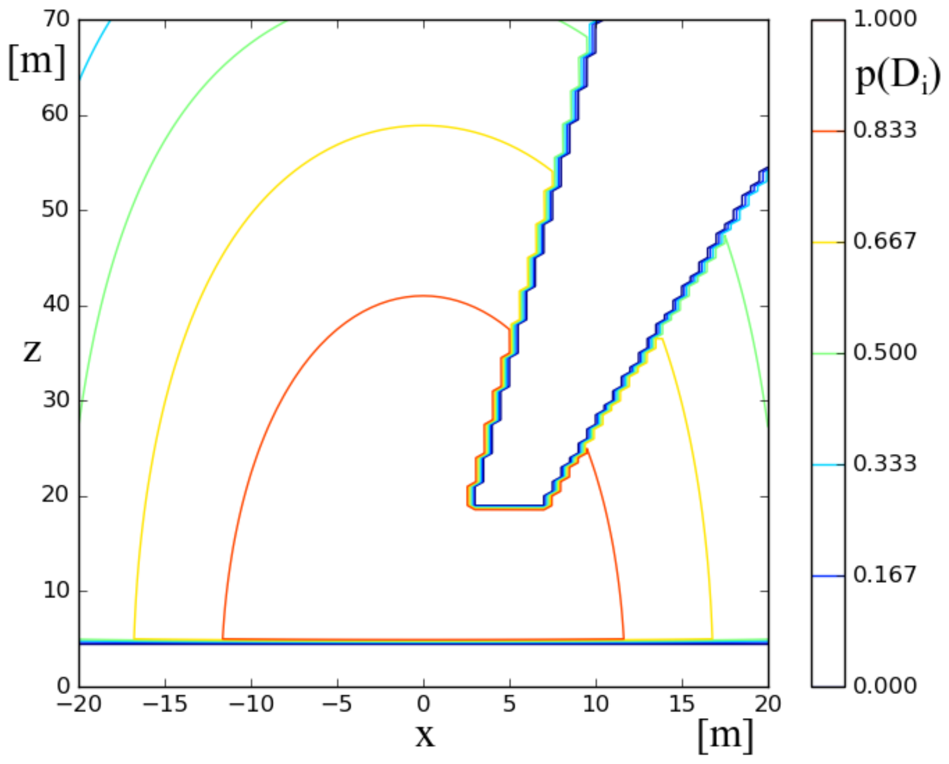


Figure 3.8: Exemplary contour plot of the direct detection contribution weight, with the sensor placed in [0,0] and an occlusion caused by another vehicle centered in [5,20]m.

visible, and that the values decay with distance and angle. It also shows the effects of an occlusion caused by a vehicle in the FOV, which makes the contribution drop to zero in the area under and behind it.

3.6 Conclusions

In this chapter we presented the mathematical formulation of a framework to compute road probability in a spatial representation, as well as the hypothesis under which it is conceived. It uses a novel indirect detection method, based on the observed and predicted behavior of the other vehicles in the scene (which we call "Behavior interpretation"), in combination with a direct detection method. Behavior interpretation analyzes the past, present and future motion of vehicles and enables our framework to exploit part of the scene understanding of other vehicles, interpreting their motion not only to infer where road is, but also to infer where it is not. The framework is formulated in a way to be independent of the implementation of the function used to estimate the vehicles behavior, and of the direct detection method used. In following chapters we will present the setup of the actual system we use for the experiments, along with the direct detection method we have chosen. The experiments show that the system has good performances on real world data. In the next chapter we will enhance the representation obtained with our framework by adding lanes and driving directions. We will present a geometric approach to divide a road representation into segments and lanes, and we will employ a behavior-based method to assign a driving direction to each lane, further exploiting the behavior of the other vehicles in the scene. The addition of this semantic information is potentially helpful in many ADAS applications.

Chapter 4

Lane Segmentation and Directions

Chapter overview *In this chapter we present a geometry-based approach for estimating semantics of road segments, given a spatial road representation. We use estimations of width and orientation, along with the motion of vehicles in the scene, to segment lanes and assign directions to them*

4.1 Motivation

Knowing the spatial layout of the road is important to act in a traffic environment, but in many cases it is not sufficient for many ADAS or autonomous driving purposes. Many applications additionally require knowledge about different road semantics. These are higher-level information that go beyond the road/non-road dichotomy. The semantics that can be of interest for ADAS applications are diverse. For example, knowledge of lane layout is necessary to predict lane change maneuvers by other drivers, and the collision risk associated with it. Speed limit information is crucial for any autonomous driving vehicle, and it depends on each particular road and lane. Emergency lanes on highways, bicycle lanes in inner city are all examples of road that should not be traversed, despite being part of road area and unoccupied. Furthermore, lanes often have only one driving direction allowed on them, so while planning maneuvers a system should make sure the ego-vehicle direction remains consistent to the one allowed at all times.

In this chapter we present an approach to estimate two of those semantic features: lane layout and driving directions. The approach uses geometric assumptions to estimate width and orientation of a road segment. It is based on a simple parallel boundaries road model, where width and orientation are related quantities. The estimation of both is carried out for every road patch, creating a distributed measure. Subsequently, patches that are close together and have a similar associated road width and orientation are clustered to form road segments. Each road segment is then divided into different lanes based on its width. Finally, the system uses the observed and estimated behavior of other traffic participants to infer the driving direction of each lane. Therefore, it can be classified as an indirect detection method, since it does not directly detect lane

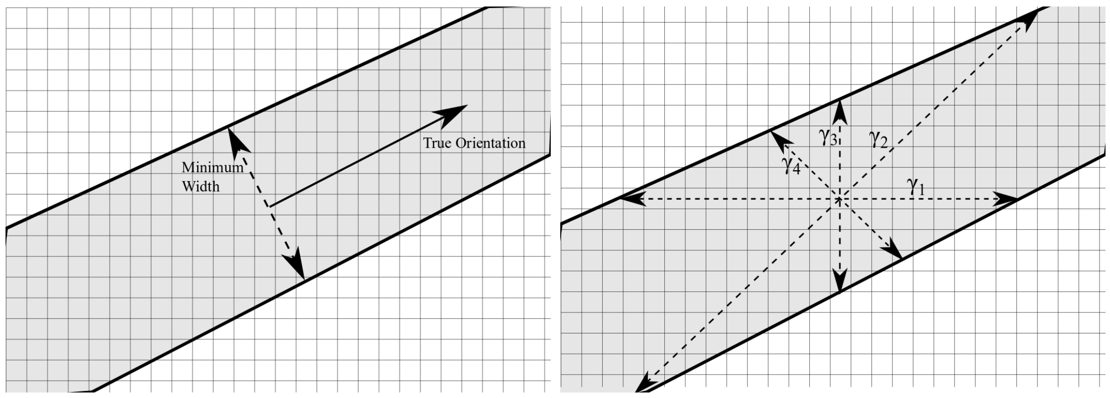


Figure 4.1: Parallel road boundaries model. The road orientation is orthogonal to its width (left). We estimate true width and orientation by sampling a subset of predefined orientations (right).

markings or driving direction signals. The approach can be applied to any grid-based spatial representation that provides a binary road classification, i.e. a patch is either road or not. As the representation we detailed in the previous chapter is probabilistic, we can always convert the result into a binary classification by applying a threshold to the road probability. If the probability is higher than the threshold, the patch is considered road and vice-versa.

4.2 Orientation and Width

As a first step we estimate the direction and width of roads. Since our goal is to perform this estimation without sensing road marking directly, we have to make geometric assumptions for our road model. The fundamental assumption of our model is that the true orientation of a road is orthogonal to its width line (defined as the shortest distance segment between its two boundaries, figure 4.1). This assumption arises from the basic model of a road with parallel boundaries. Since actual roads are not always like that, our approach will be carried out in a distributed fashion, computing local road width and orientation on each patch that belongs to road. Here, the patch-centered road width is defined as the width line passing through the patch itself, and the patch-centered road orientation is the orientation orthogonal to it.

4.2.1 Width Sampling

For every patch i that is considered road, the system estimates road orientation $\widehat{\gamma}_i$ and road width $\widehat{\omega}_i$. The algorithm starts by measuring the width of the patch-centered distance between road boundaries along a limited number of orientations (see figure 4.1), that we define using the angles γ_n in a predefined set Γ, with $0 \leq \gamma_n < \pi$. For each orientation, the width centered on patch i, $\omega_i(\gamma_n)$, is computed by scanning the representation in both directions along the selected orientation until a non-road patch is found, and then computing the euclidean distance $d_{i1,i2}$ between the last road patches in both directions. If we call those two patches l and r, we can write:

$$\omega_i(\gamma_n) = d_{i,l} + d_{i,r} \tag{4.1}$$

Additionally, we also need to compute the relative position of patch i within the road for each orientation. In order to do so, we define the value $\rho_i(\gamma_n)$ as:

$$\rho_i(\gamma_n) = \frac{d_{i,l}}{d_{i,r} + d_{i,l}} \tag{4.2}$$

Where patch l is the last road patch in the same direction defined by angle γ_n, while patch r is the last road patch in the opposite direction, $\gamma_n + \pi$.

Each possible orientation angle $\gamma \in [0, \pi)$ delivers one width value by construction. Therefore, the road width can be seen as a function of the orientation angle. According to our hypothesis, the true road orientation is the one that delivers the minimum road width:

$$\begin{aligned} &\omega_i \longrightarrow \omega_i(\gamma) \\ &\widehat{\omega}_i = min\left[\omega_i(\gamma)\right] \\ &\widehat{\gamma}_i = argmin\left[\omega_i(\gamma)\right] \end{aligned} \tag{4.3}$$

The function $\omega_i(\gamma)$ is defined for every $\gamma \in \Re$, and is periodic so that $\omega_i(\gamma) = \omega_i(\gamma + \pi)$. The road width values measured by the system (with $\gamma_n \in \Gamma$) are samples of this function. The higher the number of width values measured, the higher the accuracy of the estimation, as well as the computational time. Since the true function $\omega_i(\gamma)$ has a shape which depends on the actual shape of each road (so it could be different in different patches), we estimate the minimum by a generic second-order interpolation: the system fits a parabola to the values of γ_n at the lowest computed $\omega_i(\gamma_n)$, and its two neighbors.

$$\begin{aligned} &\widehat{\gamma}_{n,i} = argmin\left[\omega_i(\gamma_n)\right] \text{ for } \gamma_n \in \Gamma \\ &\begin{cases} \omega_i\left(\widehat{\gamma}_{n,i}\right) & = A\widehat{\gamma}_{n,i}^2 + B\widehat{\gamma}_{n,i} + C \\ \omega_i\left(\widehat{\gamma}_{n,i} + \frac{\pi}{4}\right) & = A\left(\widehat{\gamma}_{n,i} + \frac{\pi}{4}\right)^2 + B\left(\widehat{\gamma}_{n,i} + \frac{\pi}{4}\right) + C \\ \omega_i\left(\widehat{\gamma}_{n,i} - \frac{\pi}{4}\right) & = A\left(\widehat{\gamma}_{n,i} - \frac{\pi}{4}\right)^2 + B\left(\widehat{\gamma}_{n,i} - \frac{\pi}{4}\right) + C \end{cases} \end{aligned} \tag{4.4}$$

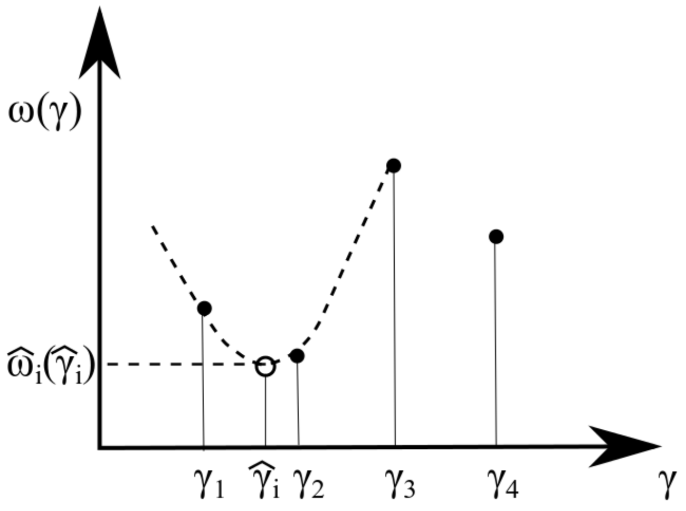


Figure 4.2: Exemplary width samples, interpolated to estimate the function minimum.

After solving the linear system and obtaining the parameters of the parabola A, B, C, finding the minimum is:

$$\begin{aligned}\widehat{\gamma}_i &= \frac{-B}{2A} \\ \widehat{\omega}_i &= C - \frac{B^2}{4A}\end{aligned} \tag{4.5}$$

The relative position ρ_i of the patch along the road is obtained with a similar concept, considering it as a function of γ. In a parallel boundaries model, we can assume the relative position of a patch is constant with respect to γ. Having measured four values of $\rho_i(\gamma_{n,i})$, we just linearly interpolate the two values that are closest to $\widehat{\gamma}_i$, i.e. $\gamma_{A,i}$ and $\gamma_{B,i}$ and estimate the relative position in correspondence of $\widehat{\gamma}_i$.

$$\widehat{\rho}_i(\widehat{\gamma}_i) = [\rho_i(\gamma_{B,i}) - \rho_i(\gamma_{A,i})] \frac{\widehat{\gamma}_i - \gamma_{A,i}}{\gamma_{B,i} - \gamma_{A,i}} + \rho_i(\gamma_{A,i}) \tag{4.6}$$

4.2.2 Model Limitations

The assumption of a straight road with parallel boundaries does not hold inside intersections. Here, the geometry of the road is completely different from the original model, and finding a road orientation and road width for each patch is an ill-posed problem. However, we could still apply our algorithm to scenes where intersections are present, in order to see how the system behaves there. Figures 4.3a and 4.3b show the output of an ideal 90 deg intersection, made by two orthogonal 10m wide roads. The road orientations inside the intersection seem inconsistent, showing a large variety of unrelated values changing abruptly in contiguous patches, while the road widths appear much larger than expected, approximately twice as large as the roads themselves. These features make it possible to distinguish and segment intersections in post-processing. Although the orientation and width values are not usable, the system currently is at least capable of recognizing intersections.

Another issue occurs in correspondence of curves in wide roads, as shown in figures 4.3c and 4.3d, where the approach is applied to an ideal circular road. In patches at the outer part of wide curves it can happen that one of the orientations next to the one with minimum width presents a very large width, due to the road curvature. It occurs when $\omega_i\left(\widehat{\gamma}_{n\pm1,i}\right) >> \omega_i\left(\widehat{\gamma}_{n,i}\right)$. This makes the second-order interpolation unsuited to find a reasonable minimum, often resulting in a value of $\widehat{\omega}_i$ that is much lower than the true width (sometimes even lower than zero). Since it depends on low sample size, this problem can be dealt with by increasing the number of sample orientations, at the cost of a longer computational time.

4.3 Road Segmentation

The next step is road segmentation. Each road patch is clustered together with other road patches, based on their width, orientation and spatial proximity. Two patches are considered part of the same road segment S_j if they are not more than 1m apart, the difference between their road width is less than a threshold m and the difference between their orientation is less than a threshold o.

4.3.1 Lane Segmentation

After clustering, the system computes the width of each road segment ω_j by averaging over the values of all its N_j patches:

$$\omega_j = \frac{1}{N_j}\sum_{i\in S_j}^{N_j}\widehat{\omega}_i \tag{4.7}$$

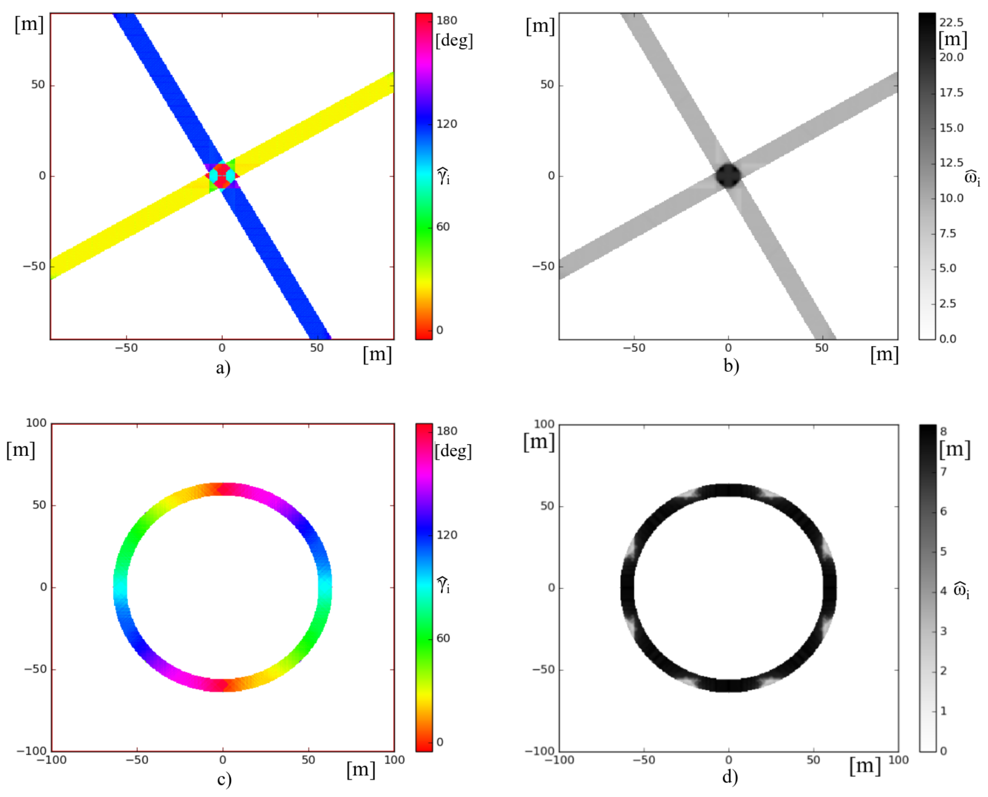


Figure 4.3: (a)(b): Output of the orientation and width estimation, performed on a 90 deg intersection made by two 10m wide roads. (c)(d): Output of the orientation and width estimation, performed on a circular road with radius 60m and width 8m. The estimations have been performed using four sample orientations (see chapter 5 for details).

Subsequently, it assigns a certain number of lanes Λ_j to each segment, based on its width and a predefined average width Ω of a lane. This value can be different in different areas or environments, and can be estimated from data statistics.

$$\Lambda_j = \lfloor \frac{\omega_j}{\Omega} \rfloor \tag{4.8}$$

After the number of lanes is determined, the system divides each road segment into the assigned number of lanes with equal width. This step is performed by checking the relative position of each patch in the road and assigning an ID to it, depending on the lane it falls into. The system assigns ID 0 to the rightmost lane, assuming the driving direction is equal to its orientation $\widehat{\gamma}_i$. This direction is set as the default direction $\overline{\phi}_i$ of each patch in the lane. All other lane IDs are assigned incrementally towards the left. The system assumes each road segment as two-way, and assigns a default direction $\overline{\phi}_i = \widehat{\gamma}_i$ to all the patches belonging to the right half of the road segment (in case of odd number of lanes, the central lane is included), and assigns a default direction $\overline{\phi}_i = \widehat{\gamma}_i + \pi$ to the patches belonging to the left half.

4.3.2 Driving Directions

The final step involves using the observed and estimated motion of all vehicles in the scene. Since our framework already has that information (see chapter 3 for details), we just need to extend that approach. Every patch i of our road representation holds a list with all the vehicles j in the scene that have a probability $p(C_{i,j})$ to traverse it in the future, present or past. For this method we need an additional information: the direction $\phi_{i,j}$ that the vehicle j had or would have (assuming a regular motion) on patch i. $\phi_{i,j}$ is estimated as follows:

$$\phi_{i,j} = \begin{cases} \phi_j & \text{if } |x_{i,j}| \leq \eta_j \vee |z_{i,j}| \leq \frac{L_j}{2} \\ \phi_j - 2\arctan\frac{x_{i,j}}{z_{i,j}} & \text{otherwise.} \end{cases} \tag{4.9}$$

Where ϕ_j is the current direction of vehicle j.

Figure 4.4 shows an exemplary plot of the direction of a vehicle centered in (0, 0)m and heading north (that is, with direction $\frac{\pi}{2}$). The estimated direction $\phi_{i,j}$ of all cars on one patch is checked and those that have an incompatible direction with respect to the patch's orientation are discarded. The remaining directions, that are similar to the road orientation within a certain angle limit, are used to decide by means of a weighted sum whether the direction of the patch has to be inverted or not. If Δ is the set of all vehicles whose directions are consistent with the default direction of the patch, and Θ the set of all vehicles whose direction is consistent with the opposite of the default, we have:

$$\phi_i = \begin{cases} \overline{\phi}_i & \text{if } \sum_j p(C_{i,j}) - \sum_k p(C_{i,k}) \geq \Xi \\ \overline{\phi}_i + \pi & \text{if } \sum_j p(C_{i,j}) - \sum_k p(C_{i,k}) < \Xi \end{cases} \tag{4.10}$$

Where $j \in \Delta$, $k \in \Theta$, and Ξ is a threshold that acts as a bias towards the initial direction: in order to invert it, the system must have a certain amount of evidence.

Having the directions distributed over all patches can allow certain areas of a lane to have a different direction from the rest of it, depending on the motion of the traffic in the scene.

4.4 Conclusions

In this chapter we presented a method for estimating additional road semantics, such as road width, orientation, lanes layout and driving directions. While this method

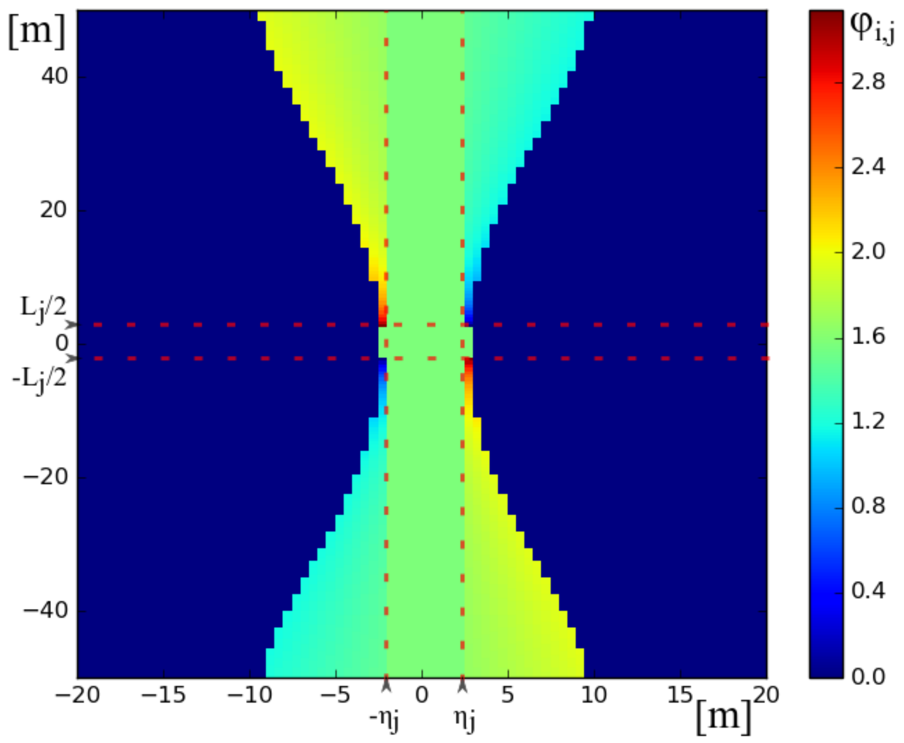


Figure 4.4: Exemplary plot of the estimated direction $\phi_{i,j}$, where j is a vehicle centered in $(0,0)$m and $\phi_j = \frac{\pi}{2}$

has been conceived with the representation presented in the previous chapter in mind, it works with any grid-based binary road representation. The approach applies geometrical considerations to infer road width and orientation in a patch-centered fashion, and then uses them to divide the road in multiple segments. Subsequently, it compares the width of each segment with a predefined average lane width to assign a number of lanes to the segment. Each lane has a default direction based on its position within the segment, and the system can change this direction based on the motion of other vehicles in the scene. Future work should focus on improving the performances of the approach in intersections, and in particular on finding a method to exploit the driving direction of other vehicles for intersection maneuvers. Furthermore, it would be interesting to investigate fusing this indirect approach with a direct one, possibly in a similar way as we did for the road/non-road framework.

The next chapter will present the concrete system architecture used for the experiments, and will detail some of the implementation of the concepts presented in these last two chapters.

Chapter 5

System Architecture

Chapter overview *In this chapter we describe the system architecture that we built around our road representation formulation to apply it to real-world data. We also specify the parameter selections that we will use for our experimental analyses in the following chapters.*

5.1 System Overview

In order to test and evaluate our theoretical formulation for road representation we constructed a system architecture around its implementation that connects with the different required data sources. Figure 5.1 shows the block diagram of the system. It can be divided into three stages. The first stage processes the inputs and computes the variables for the road probability formula. The second stage applies the formula and manages the representation grid, while the third stage uses the representation to estimate additional semantics and plan trajectories. The system has multiple inputs: the first input comprises ego-car data, from CAN and GPS, that are used to estimate the pose of the ego-car in representation coordinates (see section 5.2). The representation module uses this to transform the ego-relative coordinates of all other inputs to representation coordinates. The second input is a gray-scale image stream. The images are used by the Direct Detection module, which outputs a top-down view road confidence map (see section 5.5 for details). The third input is a list of detected vehicles with position and velocity. The detected vehicles are used by the Behavior Interpretation module, which computes the past and future positions of each of them (see section 3.3), as well as by the Occlusion Check module (see section 5.4). The latter computes the visibility of each patch, and can also use an additional input, the detected static objects. Once all inputs have been processed, the representation module computes the road probability for each patch, as explained in chapter 3. The final stage operates on the representation obtained in stage 2 with two blocks, one computing lanes and driving direction, while the other is a trajectory planner (see section 5.8), which uses the representation (with or without additional semantics) to compute the trajectory to reach a destination defined by user.

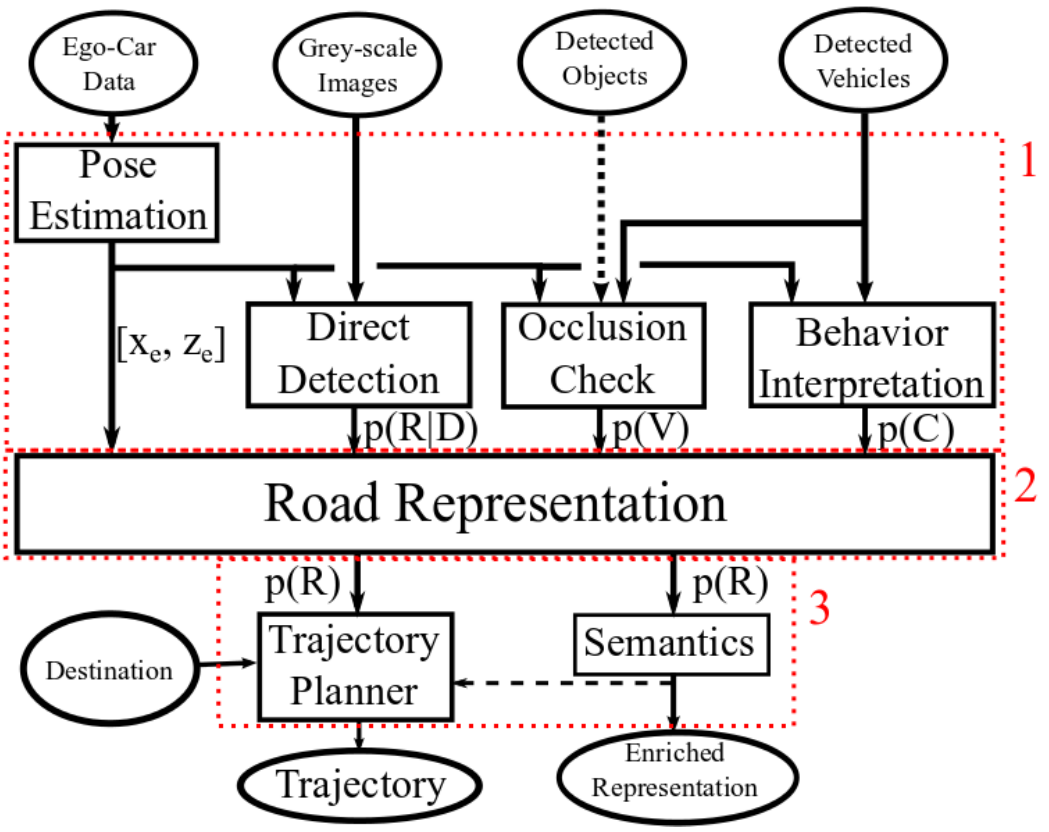
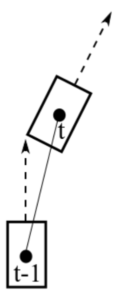


Figure 5.1: Block diagram of the system architecture. Inputs and outputs are represented as ovals, modules are rectangles. Dashed arrows represent optional connections. The three stages are highlighted in red.

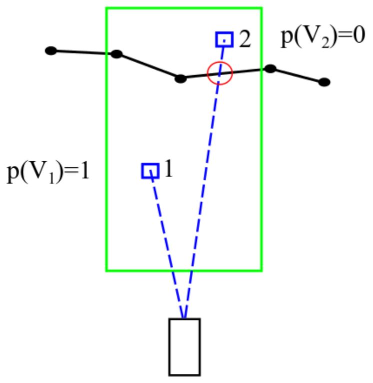
5.2 Pose Estimation

As all sensor measurements refer to the ego-car as frame of reference, we need a way to estimate its position in our representation coordinates at any point in time. Since available GPS is not reliable enough to locate the ego-car accurately, especially in inner city, we use a simplified model to estimate the motion of the ego-car, given speed measured by the sensors and orientation measured by GPS. The motion of the ego-car is approximated as linear between two iterations, so that the distance traveled depends only on its speed. If at iteration t we call the position and yaw of the ego-car respectively $\mathbf{p}_e[t]$ and $\gamma_e[t]$, and if we call ΔT the duration between two timesteps, we can compute:

$$\mathbf{p}_e[t] = \mathbf{p}_e[t-1] + \Delta T \frac{(|\mathbf{v}_e[t]| + |\mathbf{v}_e[t-1]|)}{2} \left[\cos \frac{\gamma_e[t] + \gamma_e[t-1]}{2}, \sin \frac{\gamma_e[t] + \gamma_e[t-1]}{2} \right] \tag{5.1}$$



(a) Ego-Motion estimation model



(b) Occlusion Model

Figure 5.2: (a) Sketch describing the motion estimation between two iterations $t-1$ and t. The motion is approximated as straight, with a speed and direction that are the average between the respective values measured at the two iterations. (b) Sketch describing the occlusion estimation. The black poly-line is a static object outline detected by sensors. While patch 1 is visible, patch 2 is not, as the line connecting it with the ego-car intersects the outline.

We approximate the speed and direction between iterations $t - 1$ and t as the average between the values measured at those two iterations. Figure 5.2a shows a sketch.

5.3 Vehicle Detection

The platform is equipped with 6 Ibeo Lux Lidars including Ibeo Lux Fusion System for 360 degree object detection and tracking (see figure 5.3(left)). The scanner estimates position, direction, speed and size of detected moving objects (see figure 5.3(right) as a visualization of lidar output). The Lidar software provides detected vehicles as 3D bounding boxes, and our system uses their projection onto 2D to approximate each vehicle's shape as a rectangle.

5.4 Occlusions

Detecting occlusions is necessary for a correct management of the visibility term and the balance between direct and indirect detection. In our system, occlusions are also detected by the 360 degrees Lidar scanner system mentioned in the previous section. The Lidar scanner detects two kind of objects: moving objects and static objects. We can exploit both to determine the visibility of each patch. As mentioned in section 5.3, detected vehicles are approximated as 2D rectangles by our system. Detected static objects are instead treated differently. The Lidar system only provides their outline as

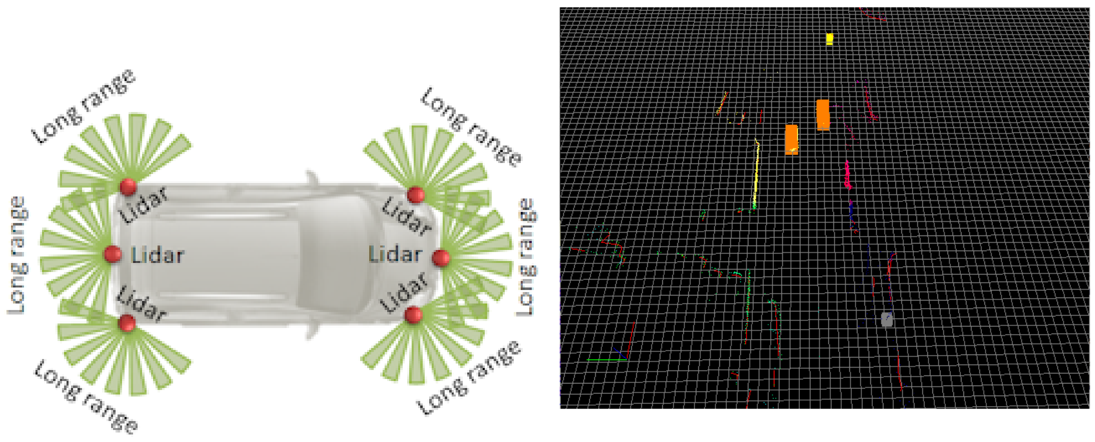


Figure 5.3: (left) The Lidar setup of our mobile platform. (right) A visual representation of Lidar output, with detected vehicles as orange boxes and static object outlines in red. Note that the grid refers to the Lidar system's own representation, and is not equal to our road representation grid.

a poly-line. The occlusion check is carried out for each patch that lies within the direct detection area. A patch is considered occluded if the line between its location and the camera location (i.e. in front of the ego-car) intersects with any static object outline or any moving object box. Figure 5.2b shows an example, with a static object outline occluding a patch from view of the ego-car. If a patch is not occluded, its visibility $p(V)$ is 1, otherwise it is 0.

5.5 Direct Detection

As mentioned in chapter 2, we chose RTDS (Fritsch et al., 2014) as direct detection system. We chose this system for its good performances in the KITTI benchmark (Geiger et al., 2012), but any kind of direct detection system providing confidence levels can be used in this framework.

RTDS takes as input a grey-scale image and, after processing it, outputs an 8-bit confidence map that is projected to Birds' Eye View (BEV) using camera perspective parameters. The result is a rectangular BEV image (with generic boundaries $x \in [x_a, x_b]$ and $z \in [z_a, z_b]$) in camera-centered coordinates, where every pixel represents confidence (from 0 to 255) that it is road (figure 5.4(left)). For our purposes, we transform the camera-centered coordinates into representation coordinates (which we can do by knowing the ego-car position, and the relative position of the camera within the ego-car) and we approximate the probability $p(R_i|D_i)$ as the confidence value in the patch, re-scaled to an appropriate range of values. The range of $[0, 1]$ can be the obvious choice, but it is not the only one. In fact, the confidence map that we use expresses "positive" road

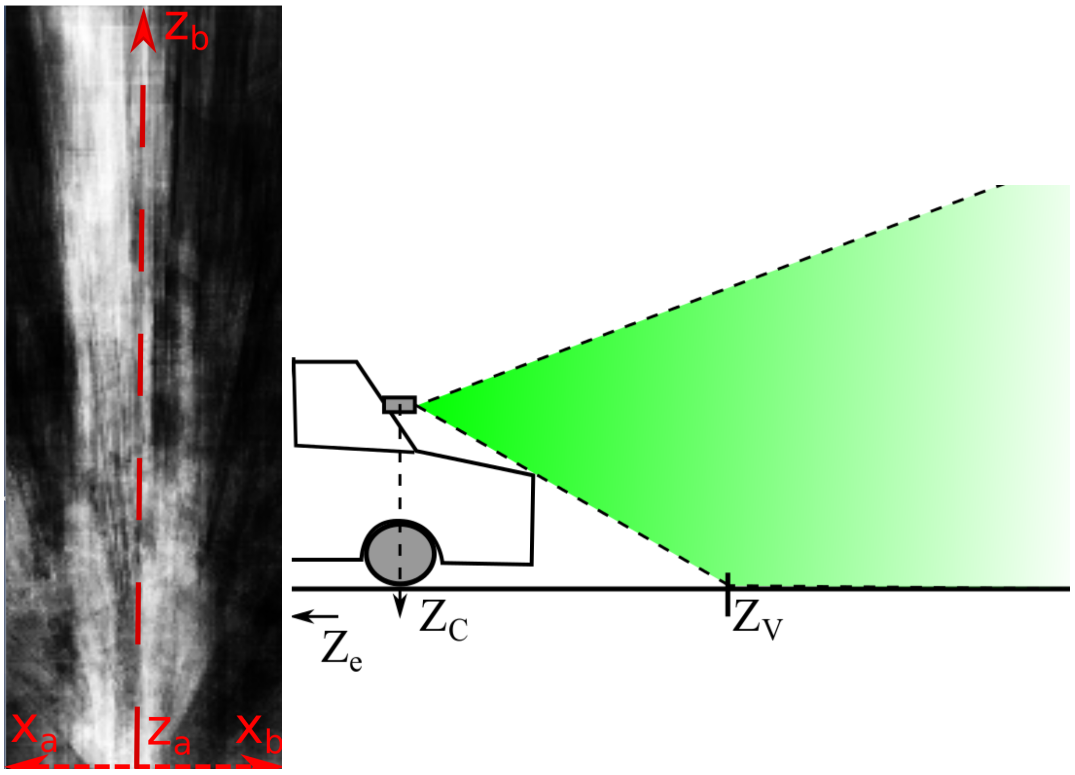


Figure 5.4: (left) Example of RTDS road confidence map, with the axis marked in red. (right) This figure shows the difference between the shortest road detection range and the camera position. The ego-car (on the far left) is considered at position Z_e (off-screen), while the camera is mounted on the front windshield, at position Z_C. The front end of the car occludes the road directly under and in front of it, so that the camera cannot see any road before point Z_V, the shortest detection range.

confidence, that is, it does not necessarily expresses a confidence between *non-road* $\leftrightarrow$ *road*, but expresses a confidence between *unknown* $\leftrightarrow$ *road*. That is why, during the translation between confidence and actual probability, the interval $[0, 1]$ might not be the most appropriate. Considering that our system implicitly uses the probability value of P_0 to express a totally unknown area (that is, in fact, the road probability value of a patch with no data available), the interval $[P_0, 1]$ might be a better choice. In our experiments we tried both (see chapter 6), with a preference for the latter.

The spatial domain of $p(R_i|D_i)$ is a rectangle (base $2b$, length $2l$) with boundaries that can be defined by the user, and should take into account the position of the camera on the car and its field of view. Ideally, the shortest detection range Z_V, which corresponds to the lower boundary of the rectangle, is determined by the camera angle and the occlusion caused by the front end of the ego-car (see figure 5.4(right)). For our experiments we have set the boundaries at $x = [-10, 10]$m and $z = [7, 62]$m, where 7m was the shortest detection range. The resulting confidence map has a resolution that can also be defined by the user, and depends on the capabilities of the camera and hardware. Setting a high resolution results in an increased computational load. For our system, we set a resolution of 0.05m, which is much finer than the one of our representation grid (0.5m). For this reason, the BEV road confidence values $\kappa(x, z)$ are averaged over each patch of our representation. In order to determine which representation patch a pixel belongs to, the location of each pixel is converted into representation coordinates and is assigned to the patch with the closest center location. After averaging over all the pixels assigned to a patch, the process provides a value $p(R_i|D_i)$ for each patch at every iteration.

$$p(R_i|D_i) = \frac{1}{N_i} \sum_{(x,z)\in i}^{N_i} \kappa(x, z) \tag{5.2}$$

In order to use these values we still need to define a reliability function. The function could be constructed based on data, by computing accurate statistics about the reliability of RTDS in the configuration used, or on modeling, as we do in this work. Our model uses a simple function, that fulfills the properties mentioned in section 3.5: it should have higher values in the middle of the Field Of View (FOV) (which in this case is a rectangle), and lower values at long distance and at the sides. The function we chose is the following:

$$p(D_i|V_i) = \cos\left(E\frac{X_{i,e}}{b}\right)\cos\left(F\frac{Z_{i,e}}{2l}\right) \tag{5.3}$$

Where the FOV has to be the same as the provided BEV confidence map, in this case being a rectangle with a width of $2b$ and a length of $2l$. $[x_{i,e}, z_{i,e}]$ is the position of patch i in the reference system of the ego-vehicle, and E, F are two constant parameters, that control the decrease in reliability along the x and z axis. In order to always obtain positive values, we have to set $E, F \in \left(0, \frac{\pi}{2}\right]$. A contour plot of this function can be seen in figure 5.5.

5.6 Representation Grid

Our approach uses a regular square grid map as an internal representation of the environment. Every patch is treated as independent from the others, and all calculations are carried out patch-wise. For this reason our representation can be implemented as a list of independent patch instances. Each patch holds several values, including its location in map coordinates. The experiments that we will present in next chapter have been using a fixed representation grid, created at the beginning of each stream with a reference

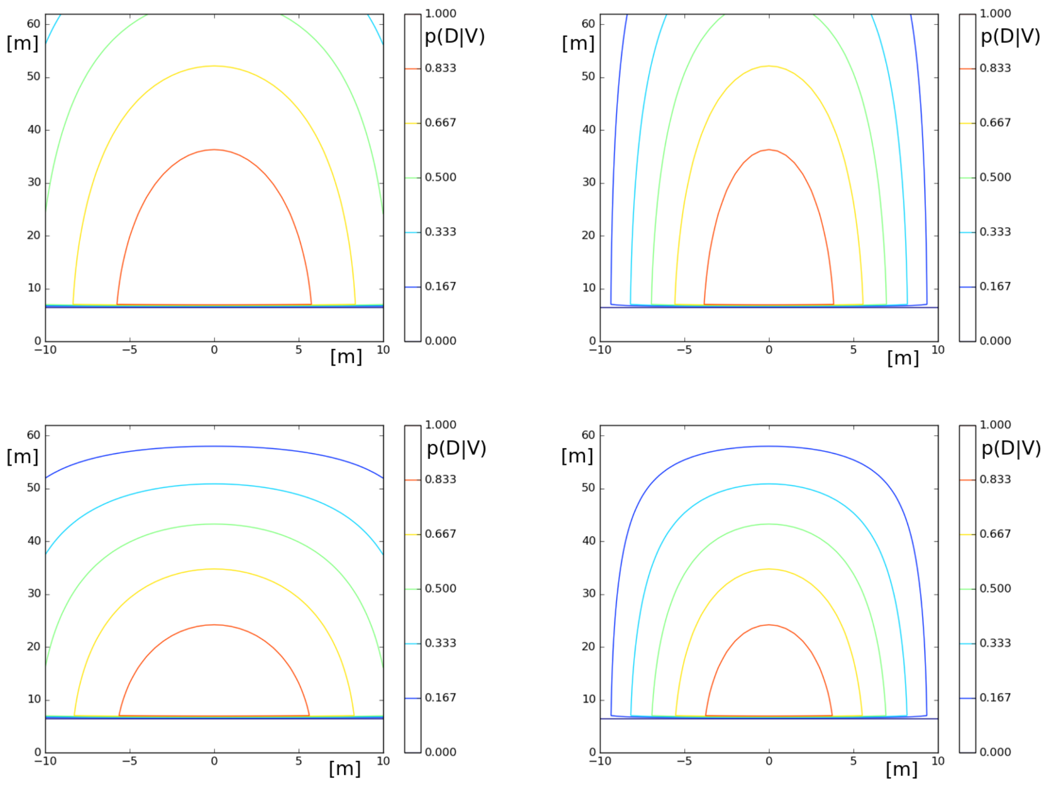


Figure 5.5: Examples of reliability function with different values of parameters E and F. (Top-Left): $E = 1$, $F = 1$. (Top-Right): $E = 1.5$, $F = 1$.(Bottom-Left): $E = 1$, $F = 1.5$.(Bottom-Right): $E = 1.5$, $F = 1.5$.

frame aligned to the ego-vehicle initial pose. Each patch of the grid represents a 0.5m x 0.5m of area. We chose this spatial resolution as a compromise between high spatial accuracy and short computational time. The representation is built at the beginning of each stream, and its coordinates are centered on to the ego-car initial position, with the z axis aligned with the ego-car initial direction (see figure 5.6).

5.7 Lanes

As explained in chapter 4, our approach needs to measure the road width along a few fixed sample orientations. For our actual implementation we used 4 orientations, defined by the angles $\Gamma = \{0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}\}$. We chose these orientations because they allow to measure the width by simply counting adjacent road patches along the horizontal, vertical and diagonal directions, and scaling the result based on the resolution chosen. In order to alleviate possible errors, orientation, width and relative position are filtered by spatial convolution with a 5x5 Gaussian mask, and finally stored into each patch.

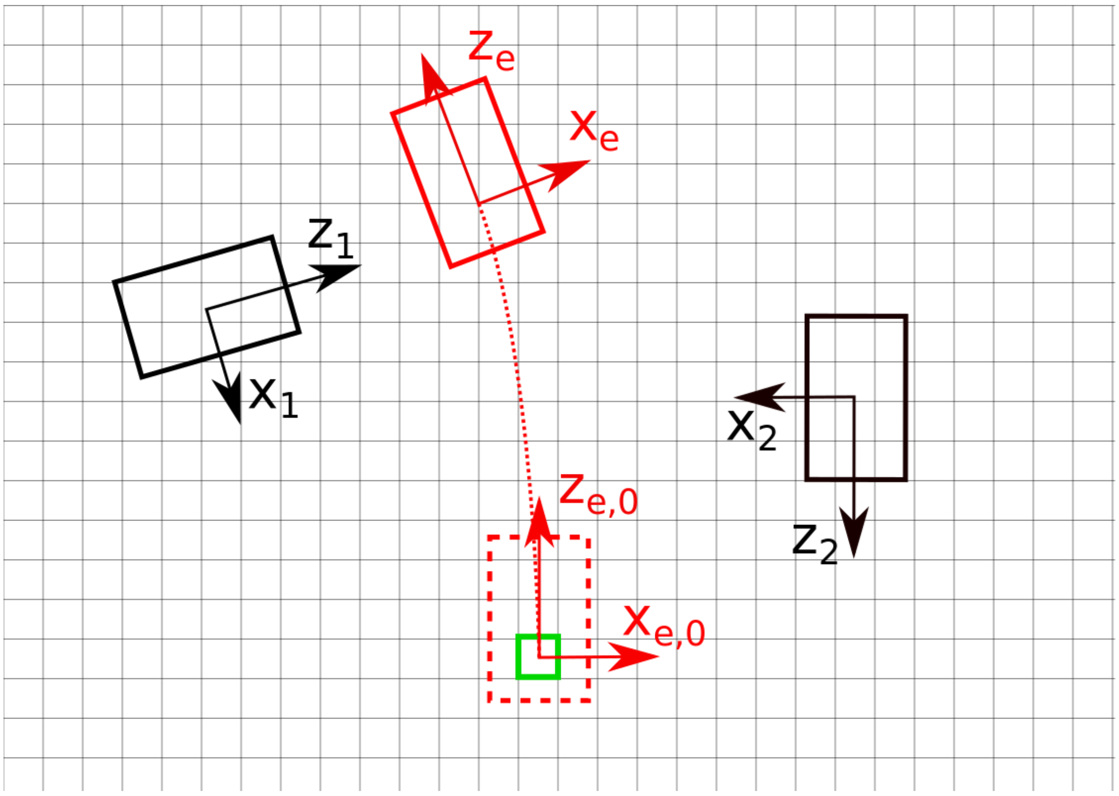


Figure 5.6: The grid is aligned with the position of the ego-car (dashed in red) at the first frame. The patch corresponding to that position (in green) is defined as patch $(0, 0)$. The system has to handle different reference frames (for example, those centered in other cars, marked in black, used for the behavior interpretation, or the current reference of the ego-car, marked in red).

The road segments are distinguished by assigning to each patch the ID of its segment. The average width that was used for the experiments presented in section 6.2 is 3.2m, which was chosen as a good compromise between typical lane widths for primary and secondary roads in Germany(Hall, Powers, Turner, Brilon, & Hall, 1995)).

5.8 Trajectory Planner

In order to show the potential of our system in terms of utility for trajectory planning, we set up a standard trajectory planner for grid representations that uses our road representation to compute trajectories. The planner employs Fringe Search (Björnsson et al., 2005), a variant of A^*, as path finding algorithm. This algorithm uses the same concept used by A^* of estimating the remaining cost to reach the target from each node in order to select which nodes to expand first, but unlike A^* it iterates over two lists, *now* and *later*, which store respectively the current and next iteration. The planner

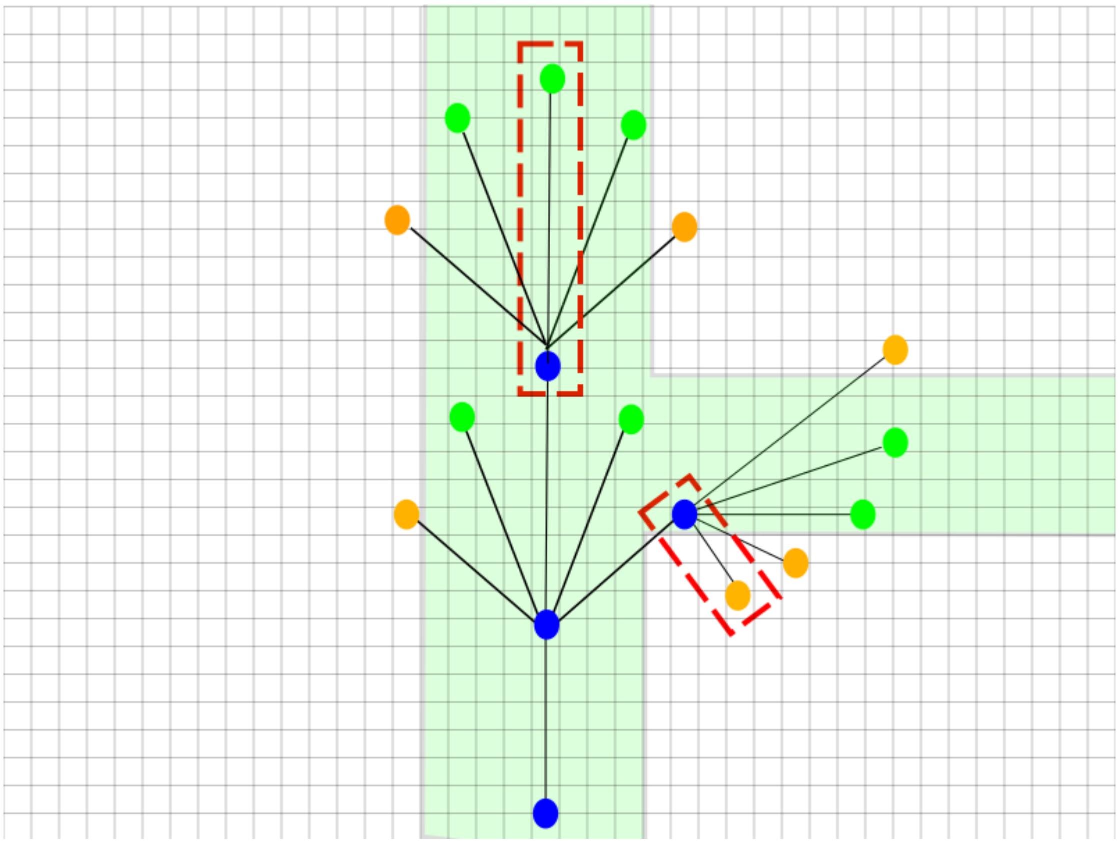


Figure 5.7: Example for a given timestep of the path planning tree structure. The cost of each node is based on all the patches within the respective red area (the width of the rectangle is the same as the ego-car). In blue are the nodes that have been expanded. In green, the nodes in *now*. In orange, the nodes in *later*. The green area represents road.

goes through the nodes stored in *now*, expands them and inserts their children in one of the two lists depending on their expected cost. If it is lower than a threshold, they get inserted in *now*, while if it is higher they go into *later*. Once the iteration over *now* ends because all the remaining nodes have an estimated cost higher than a threshold, the algorithm increases the threshold, moves all the nodes from *later* to *now*, and proceeds to the next iteration. The planner's goal is to compute the best path between the initial position A to any point in the target area. In an actual (in real time) application the target area is meant to be defined by the user, as a destination point in a navigational map. In order to model this, in our experiments the target will be an area defined by a center location decided by user. The target area has the shape of a 2D Gaussian function (that we denote f_T), where every patch has a "target score" that will be taken into account by the planner. The higher the score, the lower is the overall cost of a path ending in that point. The Gaussian shape serves to model the spatial uncertainty caused by the topological nature of a navigational map, which will have errors in the order of meters, as well as the uncertainty of GPS self-localization. The planner computes the optimal trajectory by performing a tree search, over nodes representing a certain spatial area. Such tree is exemplified in Figure 5.7. At every node the vehicle can either turn left, right or go straight. Turning is modeled as increasing the yaw rate, so that the vehicle can turn more if it started turning on the previous node. Every node can have a predefined number N_c of children, one that keeps its yaw rate and an equal number of children that either increase it or decrease it, so that $N_c = 1 + 2N_{c'}$, with $N_{c'} \in \aleph$. The number of children per node is a design choice, as a higher number means bigger coverage but longer computational time. The amount of yaw-rate increase or decrease is also a design parameter, and it depends on the dynamics of the ego-car and on the type of trajectory we want to model (e.g. harsh or smooth maneuvers). The distance $D(n)$ between a node and its parent depends on the speed of the ego-car (every node models the distance traveled by the ego-car during a fixed amount of time), which in turn is modeled as dependent on its yaw rate, in the following fashion:

$$D(n) = (V_0 - B\,|\delta_\gamma(n)|)\,T_0 \tag{5.4}$$

T_0 is a time constant that defines the time step between a node and its parent, and $\delta_\gamma(n)$ is the difference in direction between them (which models the ego-car yaw rate). The term $B\,|\delta_\gamma(n)|$, where B is a tuning parameter (strictly positive), models the slowing down of a car during a maneuver, as the distance traveled decreases with yaw rate. It also allows to have a finer tree coverage within the most important part of a maneuver.

The algorithm assigns a cost to each node, that includes a road-dependent term, a yaw-rate-dependent term and an optional, lane-dependent term. The road cost depends on the road probability of all the N patches that the ego-car will traverse if it drives there from the parent node. The same holds true for the lane-dependent cost. The formula we chose is:

$$C_R(n) = \frac{1}{N}\sum_{i\in n}^{N}\left(\frac{1}{p(R_i)} - 1\right) \tag{5.5}$$

$$C_Y(n) = |\gamma(n)| \tag{5.6}$$

$$C_\Phi(n) = \frac{1}{N}\sum_{i\in n}^{N}|\phi_e - \phi_i| \tag{5.7}$$

$$C(n) = 1 + JC_R(n) + KC_Y(n) + LC_\Phi(n) + C(n') + Y(n) \tag{5.8}$$

Here $C_R(n)$, $C_Y(n)$ and $C_\Phi(n)$ are, respectively, the cost associated to road probability, yaw rate and lane direction. Note that, in case $p(R_i)$ is lower than a certain threshold, the cost is not computed and the node is marked as a dead end. $C(n')$ is the cost of the parent node (without the estimated part), while J, K and L are tuning parameters. The estimated remaining cost $Y(n)$ to reach the target is approximated as the number of nodes required to arrive there at maximum speed. Since the base cost of each node is 1, this is the best case scenario for the cost formula, as the actual cost can never be lower than that. Every node has five children: one that maintains the yaw rate, two that increase it to the right (by 0.25 and 0.5 rad/s) and two that increase it to the left (by the same amounts). In order to speed up the algorithm, every node that has a cost higher than a predefined limit is flagged, and any of its children that also have a cost higher than the threshold are closed and marked as dead ends. The same happens if the distance to the center of target area increases for three consecutive nodes. Dead ends are eliminated by not being expanded and not being moved to the *later* list. When a path reaches a valid target, the iteration is completed and if there are multiple paths that end on a valid target the algorithm sorts them based on their cost $C(n)$ and the value of the target function at their location ($f_T(n)$), obtaining the final trajectory cost $FC(n)$.

$$FC(n) = (C_{MAX} - f_T(n))C(n) \tag{5.9}$$

Where C_{MAX} is a tuning parameter (strictly higher than 1, the maximum value of $f_T(n)$), whose value we set as 2. The trajectory with the lowest final cost is the output of the planner.

5.9 Middleware

The system has been implemented using two middlewares. The first, RTMaps™, has been used also to record the data on the mobile platform. It is a modular toolkit for stream synchronization and processing developed by Intempora (http://www.intempora.com). It allows to replay a recorded stream (camera images, Lidar-detected objects, CAN data, etc.) offline, and we use it to read and pre-process (mostly decoding) the data, as well as to visualize the inputs as they are fed to the rest of the system. The processed inputs are sent to the second middleware, ToolBOS (Brain Operating System). ToolBOS (Ceravola, Stein, & Goerick, 2008) is a brain-inspired infrastructure which focuses on modularization and synchronization of processes for intelligent systems, and comprises multiple elements. The main elements are the BBCM (Brain Bytes Component Model) and BBDM (Brain Bytes Data Model), based on which all the middleware is developed. The design tool that we used to develop our system is called DTBOS (Design Tool for Brain Operating System), while the middleware that supports its modular execution is RTBOS (Real-Time Brain Operating System). This middleware supports the execution of scripts in different languages. In our application, the scripts used were written in Python.

5.10 Conclusions

In this chapter we presented the system architecture that we used to implement our road representation and to test it on a real-world data application. The core of the system is our grid-based road representation, which we build by collecting direct detection information, from the RTDS method, and behavior-based detection information, from our own Behavior Interpretation method. The road representation can then be further enhanced with lane segmentation and driving direction, exploiting geometrical and behavioral considerations. The system also includes a trajectory planner which can compute a trajectory for the ego-vehicle to reach a user-defined destination. The planner creates a tree of spatial nodes on the representation, and employs the Fringe search algorhitm to find the path with the lowest overall cost. The system has been implemented as a set of Python scripts, using two middlewares: RTMaps for data pre-processing, and ToolBOS for the execution of RTDS and the rest of the framework. In the next chapter we will test the system on real-world data through multiple experiments.

Chapter 6

Experiments Report

Chapter overview *In this chapter we apply our framework to test its performances on real-world recordings.*

In order to evaluate the performances of our system, we recorded real world scenes with a mobile platform. The platform is a 2015 Honda CR-V equipped with 6 Ibeo Lux Lidars including Ibeo Lux Fusion System for 360 degree tracking (see figure 5.3) and an IDS UI-3580 camera in the front windshield. The composition of the dataset to be used in the evaluation is an important point. For this test to show how the approach works compared to a standard detection system, we need to test it on scenes that are relevant in terms of showing behavior interpretation effects. As such, the data have to include scenes where there are other cars on the road, as without them our representation would be equal to a direct detection system by construction, and where there are occluded roads that cannot be seen by direct detection. Considering these requirements, most of public available datasets are not suitable for our purposes. For the same reason, the dataset we test our system on is very limited in size, and for a comprehensive evaluation a much larger dataset will be needed.

6.1 Road Classification

In this section we test and evaluate the road classification performances of our system.

6.1.1 Ground Truth

The results that are shown in this section refer to a stream of over 2 minutes, part of a larger recording. In this stream our mobile platform makes a full lap around a block, occasionally stopping due to other cars. The recording was done in the city of Offenbach, Germany. In order to get a ground truth to compare our systems representation against, we hand-labeled the road area in an amount of frames necessary to get a good coverage of the whole stream, and then we built a grid map, with the same size as the representation our system will create, by mapping every labeled frame onto it. The mapping has been

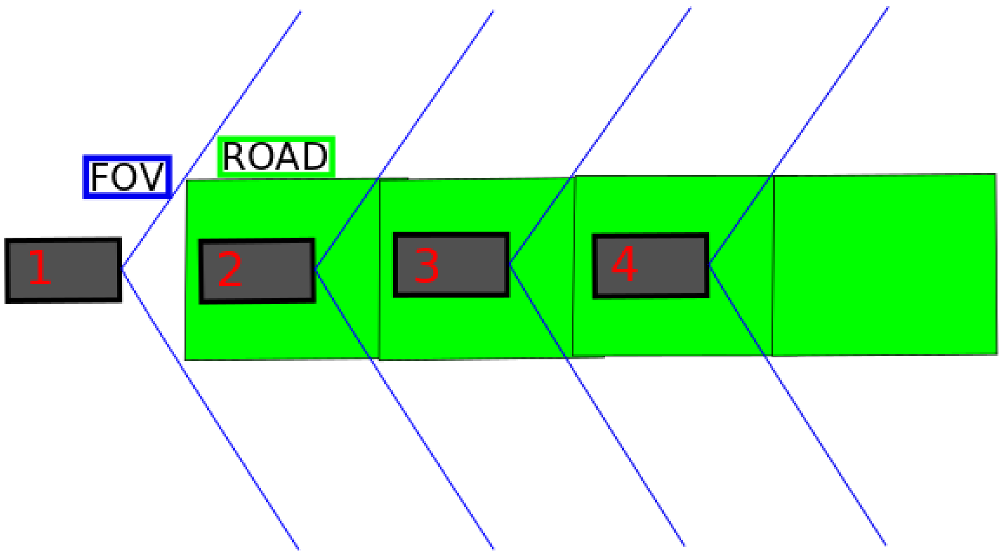


Figure 6.1: This sketch shows the process for building the ground truth. The ego-car (in grey) drives on road and an user manually annotates the road ahead (here in green), which depends on its field of view (in blue). The road annotated in different frames (here 4 frames are represented, indicated by the red number on the ego-car position) is merged together.

performed by firstly transforming every image from perspective into BEV, and then placing it with the proper position and orientation on the grid (see figure 6.1). Every square whose center location has been annotated as part of the road for at least two frames, is considered true road area. The placement onto the grid has been made by estimating the motion of the mobile platform using the approach described in section 5.2. However, the estimation of the trajectory, based on these data, is far from optimal. The reason is two-fold. On one hand, the approach used is an approximation of the actual motion of a car, and as such will always yield a certain amount of errors. On the other hand, the data (images, Lidar, CAN) used for this experiment was available at a frequency of $2Hz$. This low frequency made the ego-motion approximation very unreliable in correspondence of the sharp turns the car had to make in these narrow urban intersections, as the yaw change between frames was too large to fit into our assumption of linear motion. As a result, after loop closure the road overlap is off by

1-2m. As this method is the same we use to build our representation grid, the two representations will be properly aligned, as long as we run our tests only on the same lap we built the ground truth with, and we stop just before closure. Figure 6.2(left) shows the ground truth obtained by this process.

We chose this method for multiple reasons. First of all, comparing our representation to the ground truth frame by frame, like it is usually done for evaluating visual direct detection systems, would be limiting. This is because ground truth is affected by occlusions, as it is based on camera images, and cannot see occluded road areas, while our representation can. Road areas that lie behind occlusions are in fact an important part of our approach, and in a single frame they cannot be annotated properly and thus cannot be evaluated. Furthermore, road area that lies outside the camera field of view can be detected by our approach, as it can detect vehicles at 360-degrees, and of course that cannot be compared with a single-frame ground truth. The evaluation has been made by comparing the performances of our full system with the ones of the direct detection system it uses, the RTDS(Fritsch et al., 2014), trained on 80 images for the first stage and another 80 for the second. The images used for the training came from a different area of the same recording. Since our representation is meant to keep information from past frames, we chose to let the RTDS do so as well, by building a RTDS-only representation wherethe road probability of each patch is determined only by the average RTDS confidence over all the past frames. This was done in spite of the standard practice of evaluating the output frame-by-frame, which is usually done for direct detection systems. Taking into account the average of all past frames also enables RTDS to detect road area behind the ego-vehicle, by accumulating results from past frames.

6.1.2 Experiments

For this test we chose a representation grid of 200m x 250m, with 0.5m x 0.5m squares. The size of the representation has been chosen large enough to cover the whole block and some of the external roads. It is important to note that a few roads in the ground truth are beyond the scope of the RTDS, as the ego-vehicle never drives on them, while they can be detected by the behavior interpretation instead.

The experiment has been run by using our system with both $p(R_i|D_i)$ ranges of $[0, 1]$ and $[0.5, 1]$ (see section 5.5 for details), in order to compare the two approaches. It is important to note that the data we recorded for this experiment did not include static object outlines, so our system could only detect occlusions due to other vehicles, and not other obstacles, like buildings. This has effect on the performances of the whole system, as RTDS tends to provide many false positives during intersection maneuvers, that would be discarded (as they lie on occluded area) if we could detect buildings.

Both configurations (full system and RTDS only) have been run offline and the evaluation has been performed by comparing the output representation with the ground truth every 5 frames, sampling the threshold TH uniformly in the interval $[0, 1]$ with a step size of 0.02, for a total of 51 threshold values: a patch with a road probability higher than the threshold is considered road, while it is considered non-road in the opposite case. The classification on each of the considered frames was then compared patch-wise with the ground truth, thus building a confusion matrix. As the representation builds up with each frame, comparing it with the entire ground truth would not be meaningful. Instead, we will carry out the comparison only within a certain range around the ego-car, which is the area where the system can detect enough objects to build the representation. In fact, we will use three different ranges, to show the different spatial range of the two configurations used. The ranges that we will use are $\{30, 50, 70\}$m. We chose to evaluate the performance by computing the F1-score, which is the harmonic mean between precision and recall:

$$\begin{aligned} P &= \frac{TP}{TP + FP} \\ R &= \frac{TP}{TP + FN} \\ F1 &= \frac{2PR}{P + R} \end{aligned} \tag{6.1}$$

Where TP and FP are the numbers of true and false positives, FN is the number of false negatives, P and R are precision and recall, and $F1$ is the F1-score.

Figure 6.2 shows a comparison between ground truth and the accumulated RTDS confidence map. The accumulation has been done frame by frame, and the result is the average between all confidence values (re-scaled to $[0, 1]$) over time on each patch, without any visibility or reliability considerations. This comparison shows the strengths and weaknesses of RTDS. In particular, one can notice how many artifacts are produced around intersections, and the fact that side roads cannot be detected unless the ego-car traverses them.

Figure 6.3 shows the representation obtained by the full system, with direct detection scaled to $[0, 1]$, at a frame towards the end of the stream. The ego-car is on the right-most road section, and is about to complete the lap. We show the representation at this point, so that a few properties of it are easier to see. A few important areas have been marked with numbers to help the reader. The intersection marked with (0) is the starting position of the ego-vehicle, and thereby the origin of our representation coordinate system $(0, 0)$. The false positives in the representation are caused by mis-detections of RTDS in areas that were only seen for very few framse, during the maneuver that the ego-vehicle made in order to enter the traffic from its parking spot. From there, it makes a clock-wise lap around the whole block. The two roads marked with (1) could not be detected by a direct system at all, but the vehicles moving on them enabled the behavior interpretation

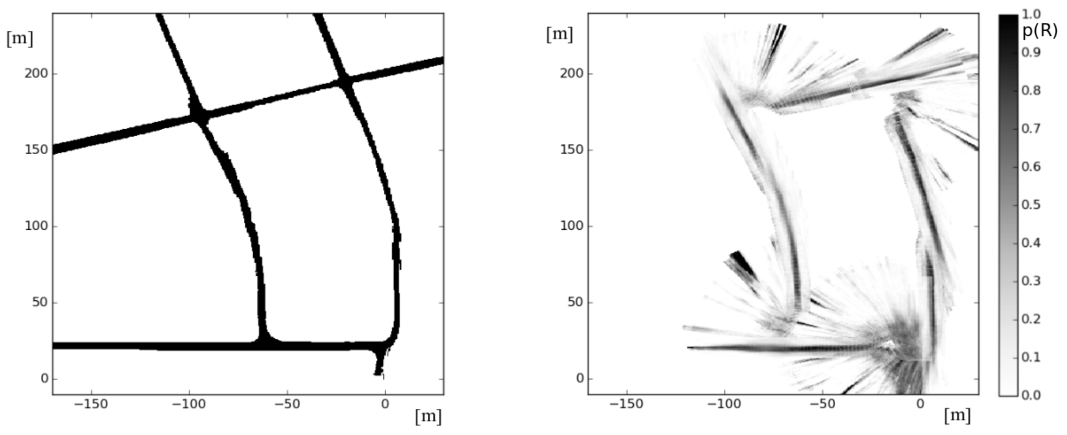


Figure 6.2: (left) The ground truth representation obtained as explained in section 6.1.1. (right) A representation obtained by accumulating the direct detection output (with the confidence values interpreted as road probability) over the whole stream.

to do so. It is also possible to see a curved road on the left side, as the vehicle that allowed us to detect it made an evasion maneuver which lowered the probability of road in the avoided area as intended. However, since the vehicle was detected by Lidar and was outside the camera FOV, we do not know the actual reason of that maneuver, and we can only assume the vehicle had to avoid an obstacle. The areas marked with (2) are artifacts caused by the RTDS, and are due to its limited training set and challenging imaging conditions. Those artifacts are mostly located on buildings, and the full system would eliminate them entirely if it could detect buildings as obstacles. Even so, however, one can notice how those artifacts are significantly filtered out by the reliability function, which gives low values at long range. The ego-car current position is marked with (3). It is possible to see that the road still has to be completed, but the RTDS is seeing part of it, and another part is given by the detection of two cars in the vicinity of the first intersection (4). One of those two cars has been detected during maneuver, and that produces the artifact that can be seen towards the right, while the other car does give us a precise idea of the road we are about to connect to. Points number (3) and (4) are the reason why we show the representation at this point in time, and not at the end of the lap, since the direct detection will eventually see the missing road, and the behavior interpretation effect would not be significant anymore to be appreciated. Marked with (5) are three intersections where the presence of vehicles in front of us allowed the behavior interpretation to partially fill the discontinuity caused by the weak performance of the RTDS, that did not detect road accurately in all intersections. The RTDS did, however, detect the external roads marked with (6), as they were clear from traffic and placed in front of the ego-vehicle. The full system also detected the side road marked with (7), although not entirely, as the vehicle traversing it was coming from an

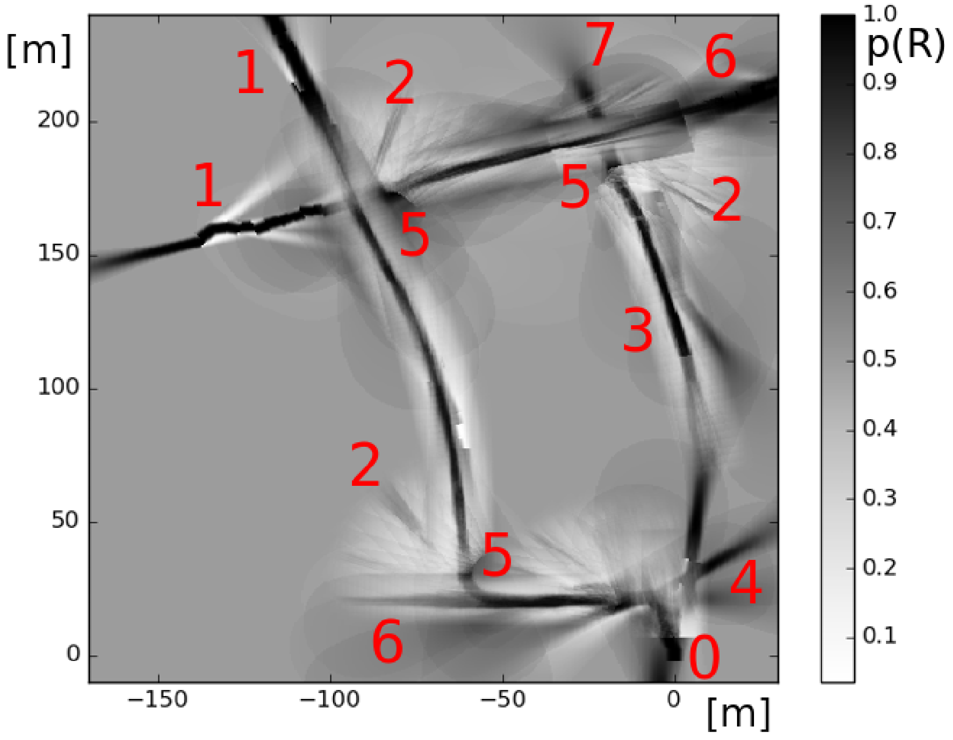


Figure 6.3: The road representation obtained by the full system (with direct detection values scaled to $[0, 1]$) towards the end of the stream.

occlusion. The figure also shows the different range of values that the full system output, compared to the "raw" direct detection: the main difference is that, while the RTDS simply gives 0 in absence of information, our full system gives 0.5. This is a potential benefit of our system, as it is not bound to a rigid classification between road and not road like RTDS, but it can classify areas as "unknown".

Figure 6.4 shows the average performances of the full system compared to the raw RTDS output over the whole stream, comparing for every frame the internal representation of each versus the ground truth only within a limited radius from the ego-car, and then averaging the F1-score over all the considered frames. From this evaluation it can be seen that the full system has a significantly higher performance than the RTDS alone. Additionally, the comparison within different ranges shows the increased spatial range of our full system, which within 70m exhibits performances superior to what RTDS does within 30m. Note that, as expected, the threshold at which the full system works best is much higher than the one of RTDS.

Figure 6.5 shows the representation achived at the end of the stream, by two different configurations of our system. On the left is the configuration that uses only direct

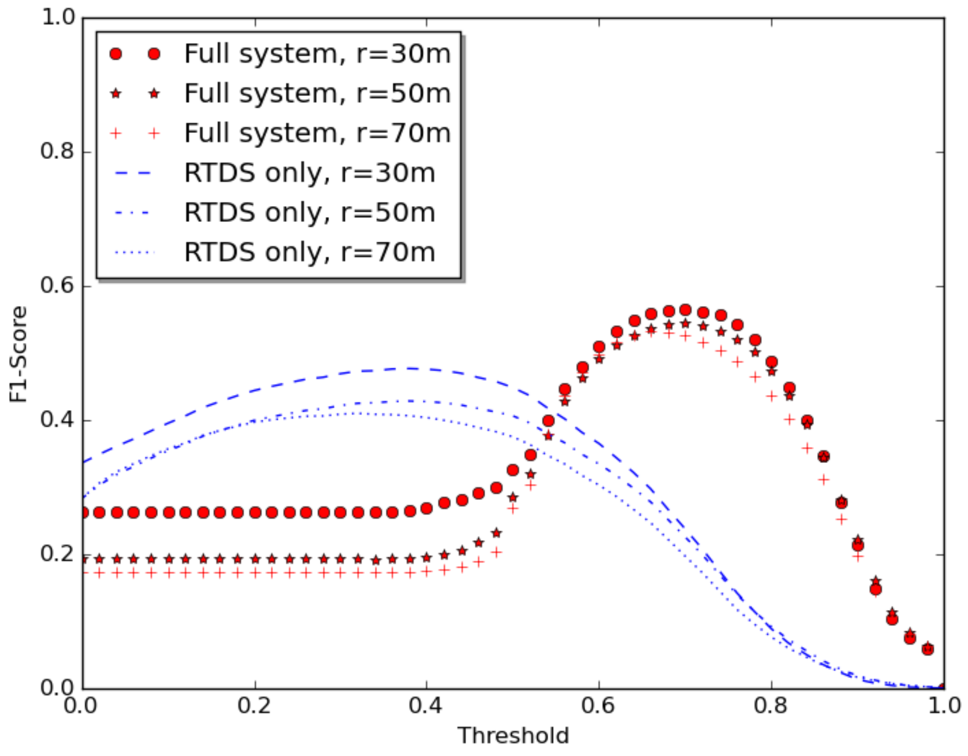


Figure 6.4: F1-score plots of the raw RTDS data and of our full system, with direct detection values scaled to $[0, 1]$. The F1-score was evaluated within three different radii around the ego-car.

detection data (including visibility and reliability) re-scaled to $[0.5, 1]$. It is basically our full system, only without behavior interpretation. It is possible to see how the reliability can partially filter out the artifacts in intersections. On the right is the full system, which uses the same direct detection data shown on the left, with behavior interpretation. Here we can see that the re-scaling of the direct detection values from $[0, 1]$ to $[0.5, 1]$ does make a significant difference qualitatively, as in this picture the transition between inside and outside the field of view appears smoother than in picture 6.3(right). Quantitatively, the effect can be seen in the evaluation shown in Figure 6.6. Here, we have evaluated the average F1-score for three configurations. The first one is the raw RTDS ouput already shown previously. The second is our system with only direct detection (scaled to $[0.5, 1]$) as shown in figure 6.5(left). The last one in our full system, with behavior intepretation, shown in figure 6.5(right). This comparison clearly shows the advantages of our framework, which takes into account the reliability of the direct detection system, and adds to it an additional source of information, the behavior of other vehicles. Note also that the addition of the behavior interpretation is more beneficial at higher ranges,

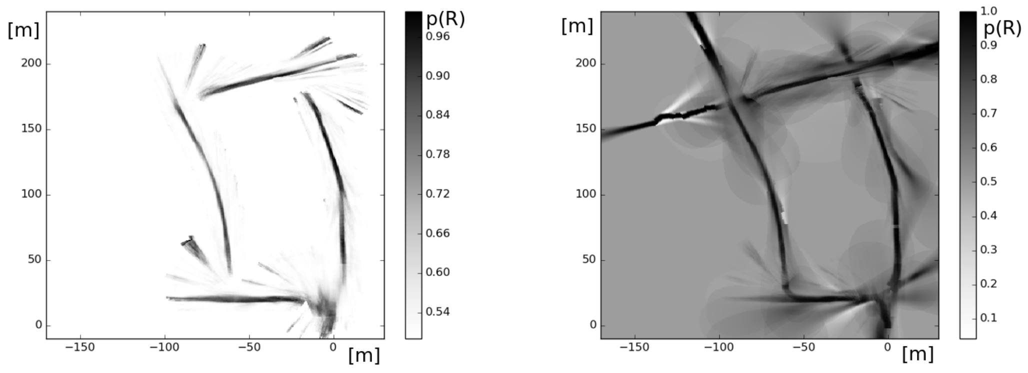


Figure 6.5: Final representation obtained by the system with two configurations, RTDS-only (left) and RTDS + Behavior interpretation (right). Both with direct detection values scaled to $[0.5, 1]$

as the increase in F1-score is higher, so that the loss in performance with distance is reduced compared to direct detection only. It can be noticed that the full system with direct detection values scaled to $[0.5, 1]$ performs slightly better than the one shown in figure 6.4, with values scaled to $[0, 1]$. Unsurprisingly, the effective threshold is higher, at around 0.75, compared to 0.7 for the former configuration.

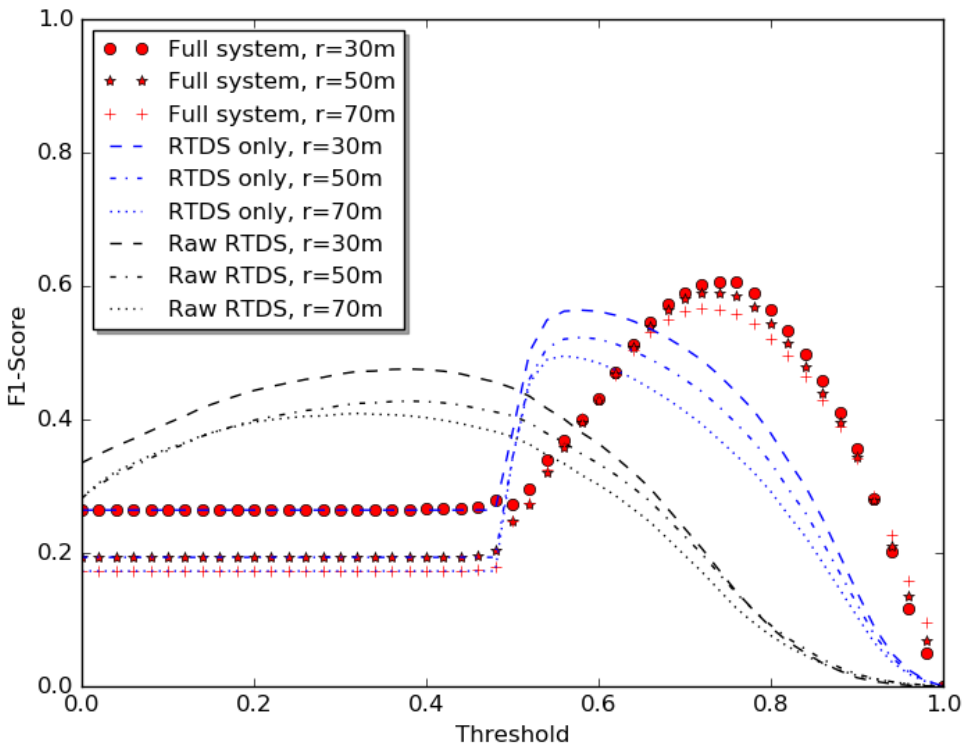


Figure 6.6: F1-score plots of the raw RTDS data and of our full system, first with only RTDS scaled to $[0.5, 1]$ and then with full behavior interpretation using the same RTDS data. The F1-score was evaluated within three different radii around the ego-car.

6.2 Lane Segmentation and Driving Directions

In order to show the potential benefits of our approach over lane segmentation and driving direction estimation, we will show a few qualitative tests on real world data. For a proper evaluation, we will need a much larger dataset. Before the real world data, however, we evaluated the geometric approach explained in section 4.2 on ideal scenes, to show the performances of the estimation, which is fundamental in our whole system. The ideal scenes are simulated road layouts (without vehicles) that have been drawn onto a representation grid with a shape defined by user.

6.2.1 Road Direction and Width Estimation

In this section we evaluate our model estimating only road orientation and road width on ideal scenes. We test our approach on two scenarios, on a 200m x 200m square grid, with patches of size 0.5m x 0.5m. The first one is a long, straight road, with a variety of different orientations and widths. Since our approach is based on measuring width along the 4 main orientations mentioned in 5.7, we want to test the capabilities of the approach on different orientations. Thus, we use the following set of orientations: 18 deg, 22.5 deg, 30 deg, 36 deg. The results (see table 6.1 and figure 6.7a-b for the 30 deg experiment, as an example) show that the approach is very robust in this particular scenario. The average error for road width was between -1.3m (the worst scenario, occurring for a 12m road with an orientation of 22.5 deg) and -0.1m (occurring for a 4m road with orientation of 36 deg). The mean square error was between 0.9m^2 and 3.3m^2. The average absolute error for road orientation ranged from a best case of less than 0.1 deg to a worst case of 1.8 deg, occurred at the widest road and the orientation of 22.5 deg, which is the most distant from any of the main orientations we use in the algorithm. The percentage of patches with an orientation error of less than 5 deg was 80% in the worst case and higher than 85% in all the others, reaching 100% in a few cases. The percentage of patches with an orientation error of less than 10 deg was always at least 99%.

The second scenario is meant to test the performances on curves. For this purpose, we generated a perfectly circular road and test our algorithm on it, using different radii and widths. By doing so, we make sure the approach is tested on all orientations. The radii we used were $40, 50, 60, 70, 80$ meters, while the road widths were $4, 6, 8, 10, 12$ meters. The results for width estimation are shown in table 6.2. This experiment clearly shows the issue discussed in section 4.2.2: the width is underestimated at the outer part of wide curves. The effect is more evident at smaller radii, and it occurs systematically at certain orientations, while at others it does not. In figure 6.7c-d one example is shown, for a width of 8m and a radius of 60m. It is apparent that the most troublesome orientations are the ones that are in between the four main orientations used by the algorithm. Over the whole experiment the average error in width was between -0.2m (occurring for all

$W \backslash \Phi$	18 deg	22.5 deg	30 deg	36 deg
4m	0.43m(11%)	0.49m(12%)	0.18m(5%)	0.09m(2%)
6m	0.65m(11%)	0.69m(11%)	0.28m(5%)	0.10m(2%)
8m	0.66m(8%)	0.90m(11%)	0.35m(4%)	0.13m(2%)
10m	0.68m(7%)	1.12m(11%)	0.46m(5%)	0.16m(2%)
12m	0.82m(7%)	1.30m(11%)	0.62m(5%)	0.20m(2%)

Table 6.1: Average absolute (and relative) error in estimated width for an ideal straight road of true width W and orientation Φ.

$W \backslash R$	40m	50m	60m	70m	80m
4m	0.44m(11%)	0.24m(6%)	0.19m(5%)	0.18m(4%)	0.20m(5%)
6m	0.81m(13%)	0.66m(11%)	0.54m(9%)	0.41m(7%)	0.30m(5%)
8m	1.27m(16%)	1.04m(13%)	0.88m(11%)	0.76m(9%)	0.62m(8%)
10m	1.55m(16%)	1.46m(15%)	1.28m(13%)	1.09m(11%)	0.94m(9%)
12m	1.80m(15%)	1.83m(15%)	1.72m(14%)	1.56m(13%)	1.33m(11%)

Table 6.2: Average absolute (and relative) error in estimated width for an ideal circular road of radius R and true width W.

narrow roads, regardless of the radius) and −1.8m (occurring for the 12m wide road and 40m radius), showing that the approach does seem slightly biased towards lower width values. As for the directions, the average absolute error was between 2 deg and 4 deg, with the worst performances occurring in the same cases as the worst width estimations. Here, the larger the radius, the better the performances. The percentage of patches with a direction error lower than 5 deg ranged from 76% in the worst situation to 97% in the best, being over 85% in 23 cases out of 25. The percentage of patches with a direction error lower than 10 deg was 94% in the worst case and was never lower than 98% in all the others, reaching 100% in multiple cases.

6.2.2 Experiments

In this section we test our system on real world data, to show the potential of our approach. The data were taken by our mobile platform, with the same setup detailed in section 6.1. As we lack a reliable way to precisely annotate the direction and width of roads, this experiments focus on lane segmentation, in particular on showing that the system can recognize the correct number of lanes (which can easily be annotated by the user), and can estimate their direction (especially with respect to the others, i.e. distinguishing opposing lanes).

In the first experiment (figure 6.8(left)), the ego-car is standing at a traffic light, on the rightmost lane of the incoming road, and observes a T-shaped intersection, made by three roads with three lanes each. In order to show the performances of the lane

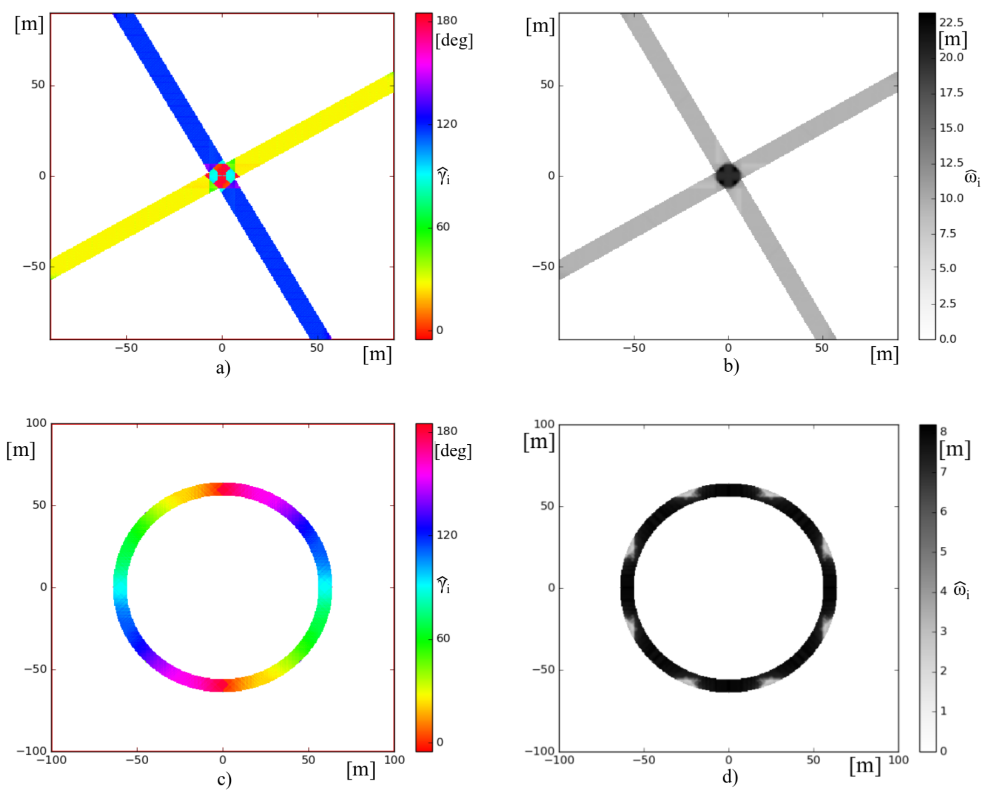


Figure 6.7: (a)(b): Output of the orientation and width estimation, performed on a 90 deg intersection made by two 10m wide roads, with an orientation of 30 deg and 120 deg. (c)(d): Output of the orientation and width estimation, performed on a circular road with radius 60m and width 8m. The estimations have been performed using four sample orientations (see chapter 5 for details).

segmentation algorithm without any influence from the rest of the system, the input road representation (figure 6.8(top-left)) has been created by manual annotations instead of by our road estimation algorithm. The Lidars detect two vehicles, both coming from the left road. One of them follows the middle lane, going straight into the right road, while the other follows the rightmost lane and turns into the same road as the ego-car stands. In figure 6.8(middle-left) it is possible to appreciate the road segmentation performed by the current system, which is not meant to handle intersections yet. Even though inside the intersection itself the segmentation is irregular, the three roads are still clearly segmented. Figure 6.8(bottom-left) shows the driving direction of each patch as inferred by the system. The system correctly divides all three road segments into three lanes and assigns the correct direction to the middle lane, from left to right, using information from the vehicle that traversed it. As for the road where the ego-vehicle is, no vehicles traverse the middle lane, and thus the system has no proper means to estimate the direction. Since the car turning into that road has a chance to be directed towards that lane, in absence of any other cue the system assigns its direction to the lane.

The second experiment (figure 6.8(right) shows a two-lane straight road, with only one direction allowed. In this experiment the input (figure 6.8(top-right)) is provided by our road representation system, thresholded at 75%, which we determined as the most effective threshold (see section 6.1.2). It has to be noted that the resulting road representation is not perfect, as it shows three small artifacts near the center line, in the form of patches incorrectly shown as non-road. These artifacts are produced due to an insufficient road visibility for RTDS (caused by bad lighting conditions and the vehicle occlusion), only partially filled by the behavior interpretation. In the stream, the ego-car travels on the left lane, and overtakes another vehicle on the right lane. Figure 6.8(middle-right) shows the lane segmentation performed by the algorithm, with the default directions assigned. Figure 6.8(bottom-right) shows the final direction of each patch. The figures shows that the system correctly divides the road into two lanes and assigns the correct direction to both of them, by observing the two vehicles in the scene. The artifacts present in the representation input do cause artifacts in lane segmentation and direction estimation, although the figures show that the effect is spatially confined.

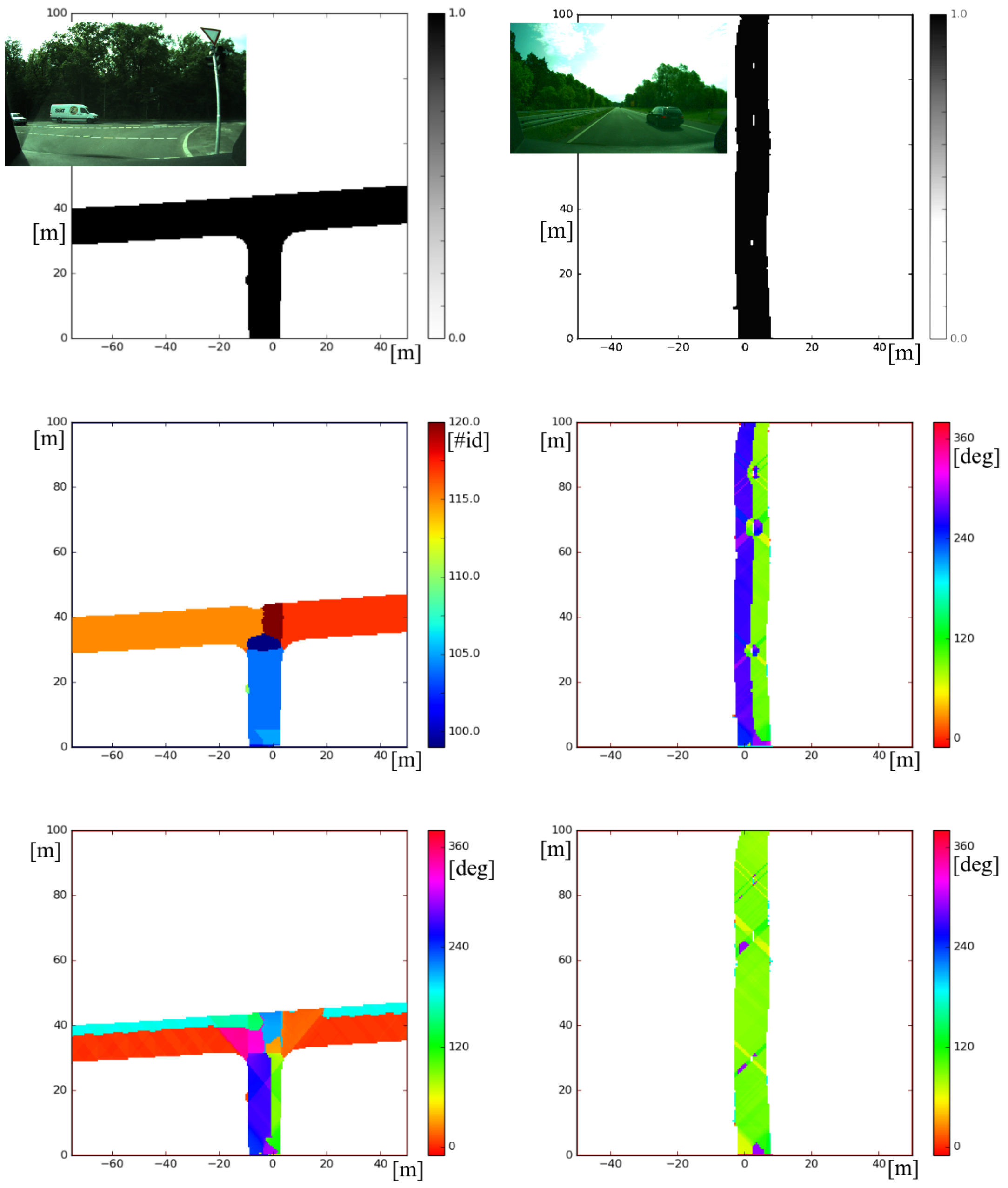


Figure 6.8: Two lane segmentation experiments side by side: (Top) Exemplary image (Top-left) and road representation input. (Middle) Road segmentation for the first experiment and default directions for the second experiment. (Bottom) Output of the lane direction estimation.

6.3 Trajectory Evaluation

In this section we apply our trajectory planner onto road representations obtained with real-world data recorded by our mobile platform in the town of Offenbach am Main, Germany. As these experiments were run using more recent data, recorded with an upgraded version of our mobile platform, they did have detected static objects outlines available, so that our system could recognize correctly areas occluded by buildings, as explained in section 5.4.

For the experiments presented in this section, the planner will not use lane and driving direction information. This allows us to decrease the complexity of the parameter evaluation in section 6.3.1 by eliminating one variable, parameter L, which will be set to 0.

Similarly to what we did in section 6.1, we will use a system with RTDS only as baseline in these experiments. Our goal is to test if the increased range granted by the behavior-based enhancement produces a representation that, by being more precise at long range and more stable over time with respect to our baseline, allows to plan more precise and stable trajectories (i.e. the computed trajectories change as little as possible over successive iterations). The experiments all use short streams, where the two representations are computed and updated at every frame of the stream. The streams chosen show intersections in inner city, with buildings occluding the destination road, since these are the best scenarios in terms of evaluating differences between our full system and the baseline. The scenes have vehicles moving from or towards the road that is defined by the user as target location. The location has been defined on a commercially available navigational map.

The planner computes the best trajectory based on each representation, and the trajectories can be compared using different measures. We focus our planning on the spatial aspect of these trajectories: the only dynamic property of the car that is taken into account is its speed, which is modeled implicitly, as it is dependent on its yaw rate. To evaluate stability over time, the most straightforward measurement is to compare the difference in placement of corresponding nodes over time. In order to establish which nodes correspond to each other, we enumerate them with an index within a trajectory, from finish to start. Nodes with the same index are considered corresponding. In case two trajectories have a different number of nodes, the nodes that have no counterpart are ignored.

However, we can extract more information from each trajectory. For example, we can define the *turning point* of a single trajectory as the location of the node that exhibits the highest yaw rate (in case there are multiple nodes with the same highest yaw rate, the turning point is the first one). We will measure both the location of the turning point and its displacement over time, to ensure that the trajectory is precise and stable. Another possible feature is the position (and stability over time) of the *end point*, which we define as the last node of a trajectory. Since the destination is defined as an area, the actual end of the maneuver can vary, and it is desirable that it moves as little as possible, to ensure a stable maneuver.

6.3.1 Parameter Evaluation

In this section we compare the results of trajectory planning with variable cost parameters J (weight on road probability cost) and K (weight on yaw rate cost), as explained in section 5.8. The goal of this experiment is to find the optimal parameters in terms of stabilityof over time of the resulting trajectories using on one hand our representation and on the other hand the baseline representation. It makes sense to find the best parameters for each of them separately, since in a real application the system would be optimized for its own representation only. In order to perform this evaluation, we set up a preliminary experiment. The system runs on a short stream with both representations computed at every frame. The trajectory planner computes the optimal trajectory to reach an area of the road to the right side of the intersection, at 4 specific frames that are separated by 0.5s from each other and starting 0.5s after the start of the stream itself. The planner uses 25 different parameters combinations (5 x 5), for a total of 200 computed trajectories. The set of combinations is limited in order to speed up the process, but for a rigorous evaluation a much larger set would be required. In order to identify the parameter set yielding the most stable results, we evaluated the obtained trajectories with three measures of stability. The first one is the standard deviation of the turning point location over the 4 computed frames of the stream. The second measure is the standard deviation of the end node, while the final measure is the average standard deviation of all corresponding nodes of the trajectories. Since all three measurements refer to a spatial displacement, establishing an importance priority between them would be difficult. We chose as the best parameter set the one that exhibits the lowest sum of all three measures (in meters). Table 6.3 shows the value for all sets. It can be noted that the sum is consistently higher for the RTDS-only configuration, which implies that its performances are overall worse (in the chosen metric) than our full system. Furthermore, note that the best parameter set for the full system has a bigger $\frac{J}{K}$ ratio, which can be interpreted as that the best performances are achieved by giving a bigger impact to the road cost, while the RTDS-only configuration prefers (in comparison) to give less impact to it, and more to the yaw-rate, probably due to the lower quality of the road representation.

6.3.2 Experiments

In this section we apply the trajectory planner to two situations using the parameters obtained in the previous section. The planner computes a trajectory on both representations obtained at 10 specific frames separated by 0.2s each. The computation of the representations started 0.5s before the first of these frames. The first stream presents a scene where the ego-car starts roughly 40m from the intersection. The target area was

	Full-System					RTDS-only				
J\ K	4	8	12	16	20	4	8	12	16	20
10	3.75	4.73	5.02	4.82	3.59	3.18	3.84	5.03	9.81	5.85
15	3.32	3.75	3.57	4.73	4.82	5.57	3.61	6.84	6.77	11.30
20	**2.81**	3.82	3.75	3.48	4.73	5.58	4.80	**3.14**	6.38	6.77
25	2.83	3.23	3.22	5.06	3.44	5.52	6.41	5.52	4.51	6.03
30	5.16	3.23	3.75	3.75	3.59	6.67	6.57	3.86	3.15	6.05

Table 6.3: Total score (sum of three measures in meters) for each combination of J and K on both configurations. Best scores highlighted in red.

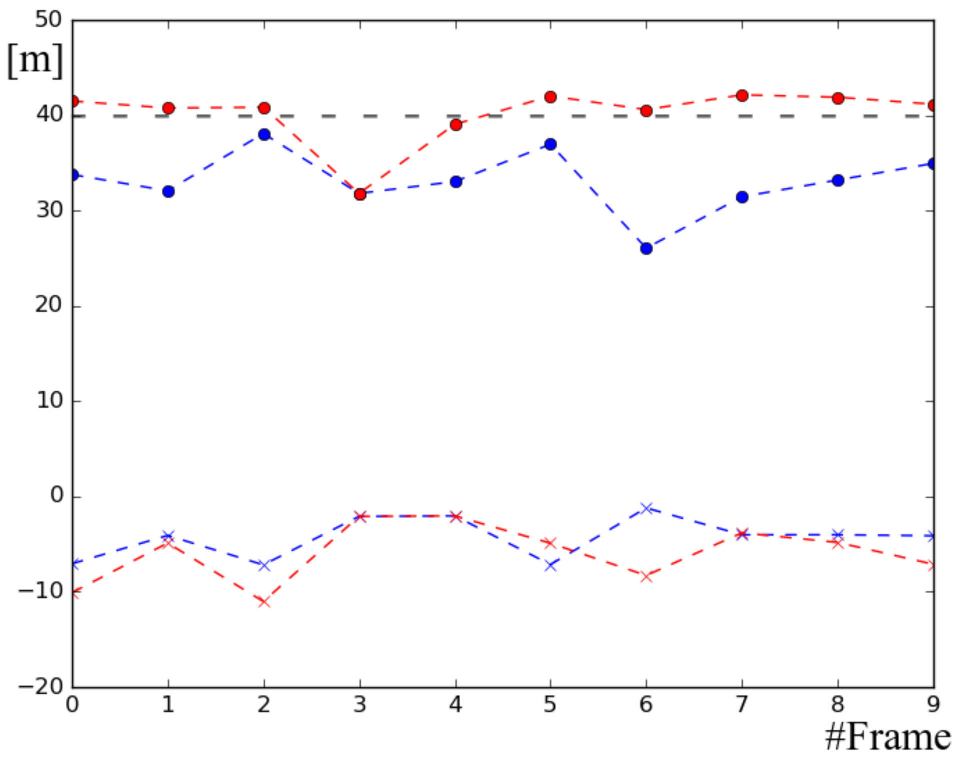


Figure 6.9: [Intersection 1] Turning point location (x coordinate as x markers, z coordinate as point markers) of the trajectories computed with the baseline representation (blue) and the full representation (red), over all 10 frames (from 0 to 9). The grey dashed line represents the true distance from the initial position to the intersection, as reference.

centered at $[-20, 38]$m with respect to the initial reference system of the ego-car, lying on the leftmost lane of a wide two-lane road (one way). Figure 6.9 shows the location of the turning point in each computed trajectory. While the standard deviation is comparable in the two configurations (3.8m for the baseline, 4m for the full representation),

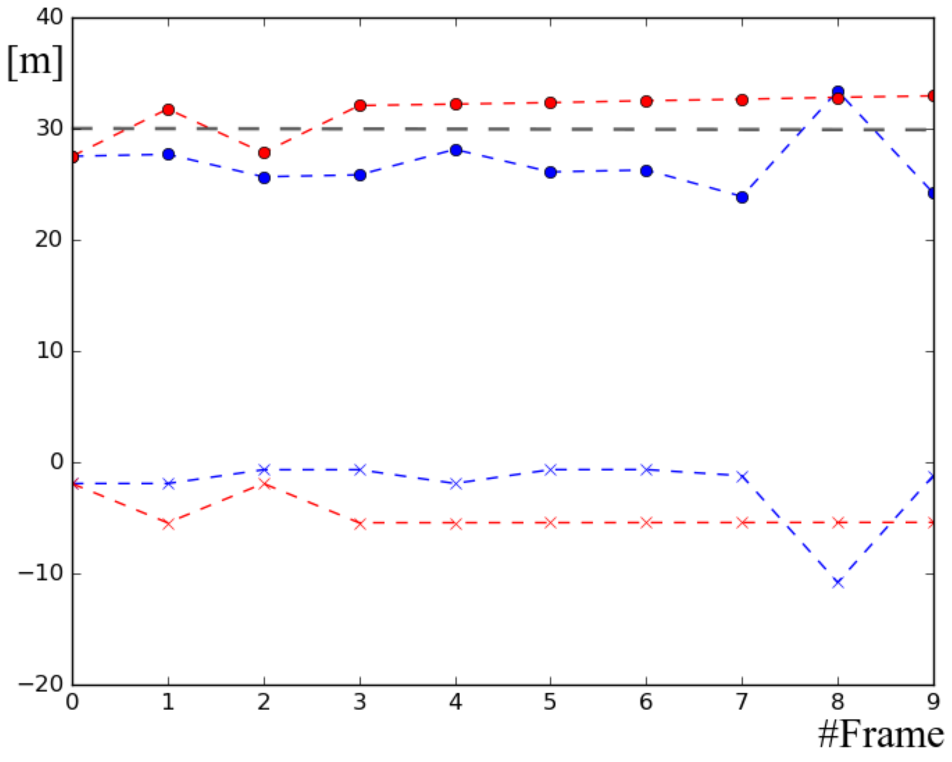


Figure 6.10: [Intersection 2] Turning point location (x coordinate as x markers, z coordinate as point markers) of the trajectories computed with the baseline representation (blue) and the full representation (red), over all 10 frames (from 0 to 9). The grey dashed line represents the true distance from the initial position to the intersection, as reference.

the average location is very different ($[-4.2, 33.1]$m for the baseline, $[-5.9, 40.2]$m for the full representation).

In Figure 6.11 we show examples of the computed trajectory for each representation in three different frames. Our full representation clearly show the destination lane, detected thanks to a vehicle traversing the intersection on it, and the planner correctly turns in the intersection. Conversely, the baseline representation cannot see the side road, and the planner turns very early, taking the shortest path in correspondence to an enlargement of the road in proximity of the intersection. The end point of both configurations is sufficiently stable, although it does show a slight advantage for our full representation (standard deviation of 0.9m against the 1.4m of the baseline configuration).

The second stream shows a similar scene, where the ego-car has to turn left to a target area centered at $[-20, 35]$m in the ego-car initial reference system,but this time the intersection is more complicated, since the target road is not perpendicular to the initial one. The intersection itself starts much closer to the ego-car than in the previous

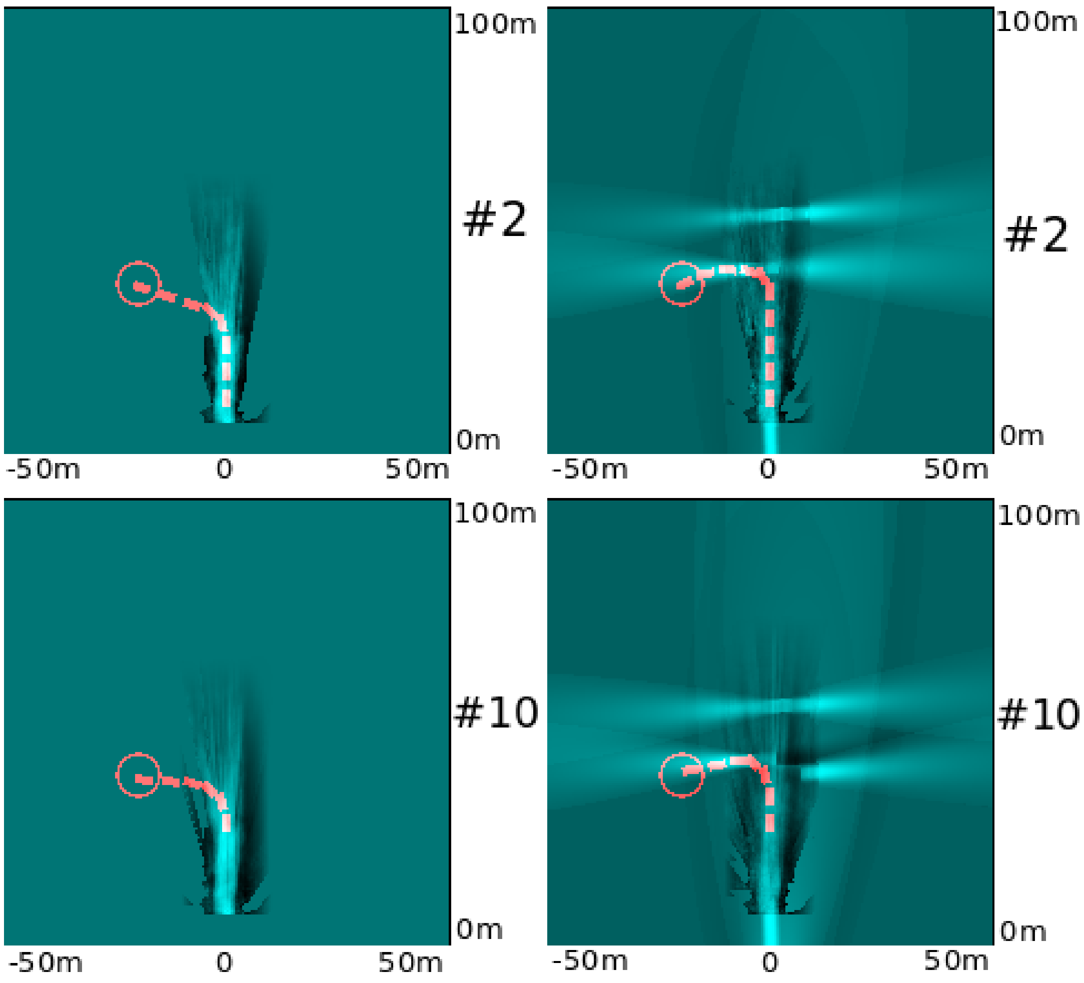


Figure 6.11: The trajectories computed with the two different configurations, (left) RTDS-only and (right) full system, at two exemplary frames (number 2 and 10) of the first intersection. The target area is marked as a red circle.

example (at around $30m$ from the initial position) as it can be clearly seen in Figure 6.12 from the road enlargement detected by RTDS. Here the two configurations show significantly different results. Although the location of the turning point shown in figure 6.10 differs less with respect to the previous example (average $[-2.2, 26.9]$m for the baseline, $[-4.7, 31.4]$m for the full representation), in this one the full representation allows for a very stable trajectory (standard deviation of 3.9m for the baseline, 2.4m for the full representation). The turning point does not change anymore after the third frame, while the baseline representation still produces unstable trajectories. The end point produced with the full representation is also much more stable than its counterpart, with a standard deviation of 0.53m against a baseline of 1.6m.

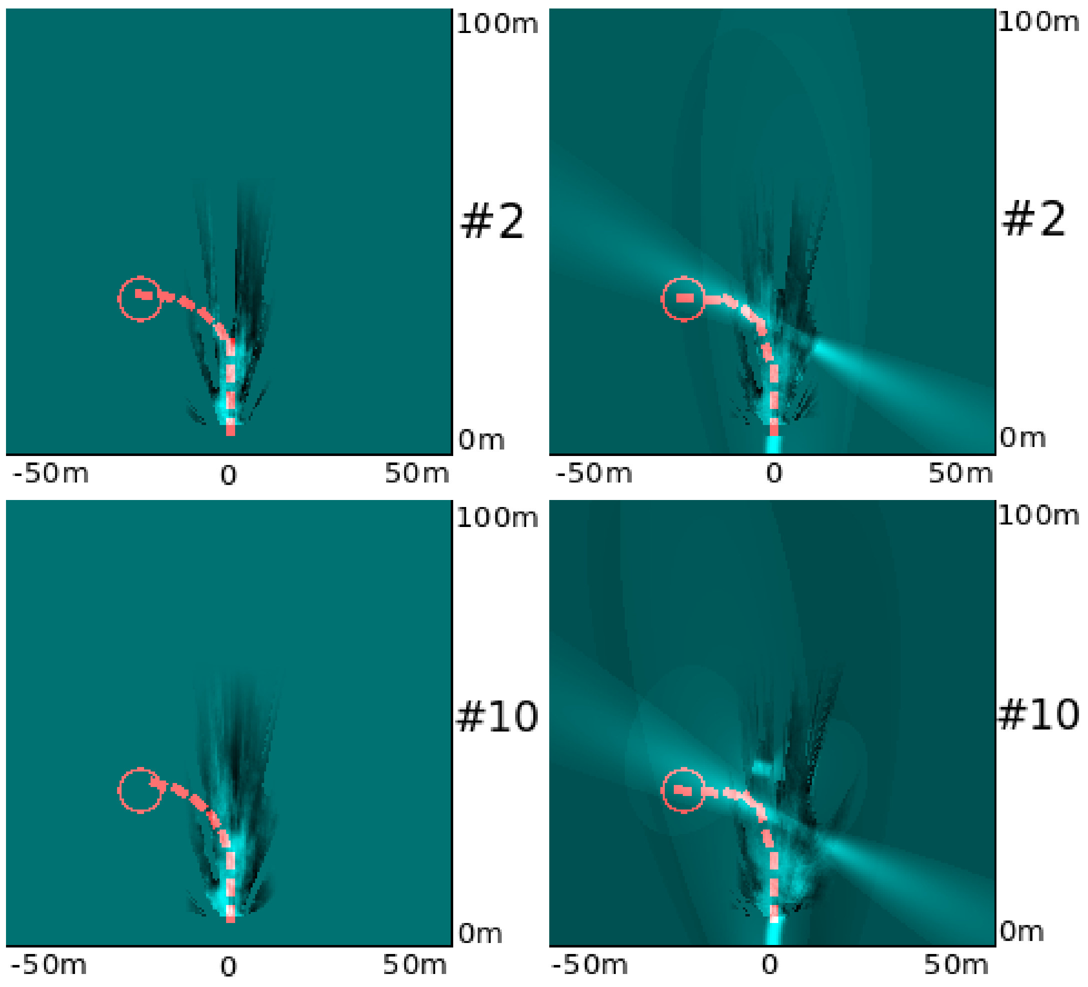


Figure 6.12: The trajectories computed with the two different configurations, (left) RTDS-only and (right) full system, at two exemplary frames (number 2 and 10) of the second intersection. The target area is marked as a red circle.

6.4 Conclusions

In this chapter we presented real-world experiments performed on different configurations of our system. We ran three types of experiments. One for evaluating the road classification performances, one for testing lane segmentation and driving directions, and one for trajectory planning. For the experiments that required it, a ground truth has been built by annotating images and merging annotated road along the stream. The experiments have been run to show the qualitative improvement in performances provided by adding behavior interpretation to a direct detection system. For this reason we compared our system to a reduced version, that does not use indirect detection, as a benchmark. The experiments showed that the classification performances of our full system outperform a purely direct detection system, particularly at long ranges (50-70m), and that this improvement in performance is relevant for trajectory planning, allowing the computation of more precise and stable trajectories. Future work should focus on creating and using a larger dataset for an extensive quantitative evaluation of the system. Unfortunately, existing datasets (e.g. KITTI) were built with direct detection in mind, and they lack scenes that are significant for our approach, which are those with other vehicles moving to occluded areas of interest. In the next chapter we will conclude by discussing the strengths and limitations of the presented approach, as well as suggesting future work to improve it.

Chapter 7

Conclusions

Road detection is a key issue for modern ADAS and autonomous driving systems, as they need a precise and reliable understanding of the road layout while operating. The motivation of this thesis was to improve the performance and reliability of state-of-the-art direct road detection systems, which suffer from various environmental issues, in particular occlusions, that are abundant in inner city scenarios. The aim was to develop a novel indirect approach for road detection, which exploits the behaviour of the other vehicles to infer road probability on a two-dimensional representation grid. Since the other vehicles are a main cause of occlusion in the aforementioned scenarios, such an approach can be expected to perform best in situations that are particularly unfavorable to direct detection systems, and therefore it can complement them. For this reason, we presented a consistent framework for fusing information from the two sources. The details of the indirect approach, as well as those of the framework, were presented in chapter 3. In chapter 4 we showed how to extend the representation obtained previously by adding additional semantics. The semantics chosen were lanes and driving directions. Our approach estimates them by employing geometrical considerations to segment roads and lanes, and subsequently by observing the behavior of the other vehicles to infer driving directions on each lane. We designed a system in which we could implement and test this approach. The system is described in chapter 5. Tests were run on real world data recorded by our mobile platform. The experiments were described in chapter 6 and can be divided into three categories, based on their goal. The first category aimed at spatially evaluating the road classification performances of the framework. The classification was compared patch-wise to a ground truth obtained by manual annotation of images accumulated over the whole stream. The second category aimed at showing qualitatively the potential of the lane and driving direction estimation. The third category aimed at using our representation to compute trajectories, employing a custom trajectory planner, showing that it allows for more precise and stable trajectories over multiple iterations. The experiment showed that the addition of indirect detection can significantly improve the road classification with respect to a baseline system using only direct detection, and that this improvement has a positive effect on the quality (in the metrics we considered) on the computed trajectories.

The concept of indirect road detection and its inclusion in a coherent framework has the potential to greatly improve road detection, particularly in inner-city scenarios, where state-of-the-art systems still struggle. The fusion of direct and indirect detection covers the weak points of both, and allows for a more reliable and stable environment understanding, especially at mid-range (50-70m).

The nature of this approach (in particular of the indirect detection part) makes it complicated to define a proper method for a full evaluation of its performances. In fact, our approach presents its best performances only in certain well defined scenarios, since it depends on the presence and number of other vehicles on the road. While this is a problem in terms of scarsity of scenes in annotated public databases (as mentioned in chapter 6), this is also a problem in terms of deciding which kind of scenes would be fair to evaluate our approach on. In fact, picking scenes full of traffic and occlusions would guarantee the best conditions for indirect detection, by employing the worst conditions for direct detection. On the other hand, picking scenes with low traffic would not show any significant improvements over a standard direct detection system. A proper evaluation would need a dataset including various different scenarios, having a realistic distribution.

Another important point for a full evaluation is the metric that will be used to evaluate the performances. In this thesis we used various methods, e.g. patch-wise comparison with ground truth. While we believe that the metrics we chose are significant enough to show the performances of our framework as a proof of concept, a full evaluation may need different metrics, or just more of them, especially when comparing trajectories.

Aside from performing an extensive evaluation, the system can also be refined in terms of employing better models. For example, the estimation of the probability $p(C)$ can use a more realistic model, possibly taking into account car dynamics, or real-world statistics. The same can be said for the direct detection reliability function, which depends on the particular direct detection method used. These refinements are possible because our framework does not depend on the particular models employed by its components. In fact, we regard its versatility as one of the best features of our approach.

Finally, indirect detection as a concept could be extended to other elements of road environment, and future research may aim to investigate upon those. For example, the state of a traffic light could be inferred by observing the behavior of other cars, while the behavior of pedestrians may be used to detect sidewalks and zebra-crossings, and the behavior of bycicles could be used to detect bycicle lanes. In general terms, the behavior of other intelligent entities is a resource that is routinely used by biological intelligence, and therefore future intelligent cars should also try to exploit it.

References

Alin, A., Butz, M. V., & Fritsch, J. (2011). Tracking moving vehicles using an advanced grid-based Bayesian filter approach. *2011 IEEE Intell. Veh. Symp.*(Iv), 466–472.

Alvarez, J. M., Lopez, A. M., Gevers, T., & Lumbreras, F. (2014). Combining Priors, Appearance, and Context for Road Detection. *IEEE Trans. Intell. Transp. Syst.*, *15*(3), 1168–1178.

Bacha, A., Bauman, C., Faruque, R., ..., Covern, D. V., & Webster, M. (2008). Odin: Team VictorTango's entry in the DARPA Urban Challenge. *J. F. Robot.*, *25*(8), 467–492.

Badino, H., Franke, U., & Mester, R. (2007). Free Space Computation Using Stochastic Occupancy Grids and Dynamic Programming. In *Iccv 2007 work. dyn. vis.* Rio de Janeiro, Brazil.

Badino, H., Franke, U., & Pfeiffer, D. (2009). The stixel world-a compact medium level representation of the 3d-world. In *Dagm-symposium* (pp. 51–60).

Bar Hillel, A., Lerner, R., Levi, D., & Raz, G. (2014). *Recent progress in road and lane detection: A survey* (Vol. 25) (No. 3). Springer Verlag.

Bender, P., Ziegler, J., & Stiller, C. (2014). Lanelets: Efficient Map Representation for Autonomous Driving. In *Ieee intell. veh. symp. (iv 2014)* (pp. 420–425). Dearborn, Michigan, USA: IEEE.

Björnsson, Y., Enzenberger, M., Holte, R. C., & Schaeffer, J. (2005). Fringe search: Beating a* at pathfinding on game maps. In *Ieee symposium on computational intelligence and games (cig05).*

Bonnin, S., Weisswange, T., Kummert, F., & Schmüdderich, J. (2014). Pedestrian Crossing Prediction using Multiple Context-based Models. In *17th int. ieee conf. intell. transp. syst.* IEEE.

Bouzouraa, M. E., & Hofmann, U. (2010). Fusion of occupancy grid mapping and model based object tracking for driver assistance systems using laser and radar sensors. In *Ieee intell. veh. symp. (iv 2010)* (pp. 294–300). San Diego, USA: IEEE.

Broggi, A., Cattani, S., Patander, M., Sabbatelli, M., & Zani, P. (2013). A full-3d voxel-based dynamic obstacle detection for urban scenario using stereo vision. In *Intelligent transportation systems-(itsc), 2013 16th international ieee conference on* (pp. 71–76).

Brust, C.-A., Sickert, S., Simon, M., Rodner, E., & Denzler, J. (2015). Convolutional patch networks with spatial prior for road detection and urban scene understanding. *arXiv preprint arXiv:1502.06344*.

Caltagirone, L., Scheidegger, S., Svensson, L., & Wahde, M. (2017, June). Fast lidar-based road detection using fully convolutional neural networks. In *2017 ieee intelligent vehicles symposium (iv)* (p. 1019-1024).

Ceravola, A., Stein, M., & Goerick, C. (2008). Researching and developing a real-time infrastructure for intelligent systems – evolution of an integrated approach. *Robotics and Autonomous Systems*, *56*(31), 14–28.

Danescu, R., Oniga, F., & Nedevschi, S. (2011). Modeling and Tracking the Driving Environment With a Particle-Based Occupancy Grid. *IEEE Trans. Intell. Transp. Syst.*, *12*(4), 1331–1342.

Dolgov, D., Thrun, S., Montemerlo, M., & Diebel, J. (2010). Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, *29*(5), 485–501.

Fatemi, M., Hammarstrand, L., Svensson, L., & García-Fernández, A. F. (2014). Road Geometry Estimation Using a Precise Clothoid Road Model and Observations of Moving Vehicles. In *17th ieee int. conf. intell. transp. syst. (itsc 2014)* (pp. 238–244). Qingdao, China: IEEE.

Ferguson, D., Howard, T. M., & Likhachev, M. (2008). Motion planning in urban environments. *Journal of Field Robotics*, *25*(11-12), 939–960.

Franke, U., Pfeiffer, D., Rabe, C., Knoeppel, C., Enzweiler, M., Stein, F., & Herrtwich, R. G. (2013). Making bertha see. In *Proc. ieee int. conf. comput. vis.* (pp. 214–221). Institute of Electrical and Electronics Engineers Inc.

Fritsch, J., Kühnl, T., & Kummert, F. (2014). Monocular Road Terrain Detection by Combining Visual and Spatial Information. *IEEE Trans. Intell. Transp. Syst.*.

Geiger, A., Lauer, M., Wojek, C., Stiller, C., & Urtasun, R. (2013). 3D Traffic Scene Understanding from Movable Platforms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 1–14.

Geiger, A., Lenz, P., & Urtasun, R. (2012). Are we ready for autonomous driving? The KITTI vision benchmark suite. In *Ieee comput. soc. conf. comput. vis. pattern recognit. (cvpr 2012)* (pp. 3354–3361). IEEE.

Gindele, T., Brechtel, S., Schröder, J., & Dillmann, R. (2009). Bayesian Occupancy Grid Filter for Dynamic Environments Using Prior Map Knowledge. , 669–676.

González, D., Pérez, J., Milanés, V., & Nashashibi, F. (2016). A review of motion planning techniques for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, *17*(4), 1135–1145.

Guo, C., Kidono, K., & Ogawa, M. (2016, June). Learning-based trajectory generation for intelligent vehicles in urban environment. In *2016 ieee intelligent vehicles symposium (iv)* (p. 1236-1241).

Guo, C., Meguro, U., Yamaguchi, K., Kidono, K., & Kojima, Y. (2014). Improved Lane

Detection based on Past Vehicle Trajectories. In *17th ieee int. conf. intell. transp. syst. (itsc 2014)* (pp. 1956–1963). IEEE.

Hall, L., Powers, R., Turner, D., Brilon, W., & Hall, J. (1995). Overview of cross section design elements. In *International symposium on highway geometric design practices.*

Hart, P. E., Nilsson, N. J., & Raphael, B. (1972). Correction to a formal basis for the heuristic determination of minimum cost paths. *ACM SIGART Bulletin*(37), 28–29.

Jacobs, G., & Aeron-Thomas, A. (2000). A review of global road accident fatalities. *Paper commissioned by the Department for International Development (UK) for the Global Road Safety Partnership.*

Jung, C., & Kelber, C. (2004). A robust linear-parabolic model for lane following. *Proceedings. 17th Brazilian Symp. Comput. Graph. Image Process.*, 72–79.

Kang, J., & Chung, M. J. (2011). Stereo-vision based free space and obstacle detection with structural and traversability analysis using probabilistic volume polar grid map. *2011 IEEE 5th Int. Conf. Robot. Autom. Mechatronics*, 245–251.

Kühnl, Tobias, Kummert, F., & Fritsch, J. (2011). Monocular road segmentation using slow feature analysis. In *Ieee intelligent vehicles symposium (iv).* Washington, USA: IEEE.

Liebner, M., Baumann, M., Klanner, F., & Stiller, C. (2012). Driver intent inference at urban intersections using the intelligent driver model. In *Ieee intell. veh. symp. (iv 2012)* (pp. 1162–1167). Alcalá de Henares, Spain: Ieee.

Likhachev, M., Ferguson, D. I., Gordon, G. J., Stentz, A., & Thrun, S. (2005). Anytime dynamic a*: An anytime, replanning algorithm. In *Icaps* (pp. 262–271).

Matthaei, R., Bagschik, G., & Maurer, M. (2014). Map-relative localization in lane-level maps for ADAS and autonomous driving. In *Ieee intell. veh. symp. proc.* (pp. 49–55). Institute of Electrical and Electronics Engineers Inc.

Miller, I., Campbell, M., Huttenlocher, D., ..., Kurdziel, M., & Fujishima, H. (2008). Team Cornell's Skynet: Robust perception and planning in an urban environment. *Journal of Field Robotics*, *25*(8), 493–527.

Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., ... others (2008). Junior: The stanford entry in the urban challenge. *Journal of field Robotics*, *25*(9), 569–597.

O'Callaghan, S. T., Singh, S. P. N., Alempijevic, A., & Ramos, F. T. (2011). Learning navigational maps by observing human motion patterns. In *Ieee int. conf. robot. autom. (icra 2011)* (pp. 4333–4340). Shanghai, China: IEEE.

Petrovskaya, A., & Thrun, S. (2009). Model based vehicle detection and tracking for autonomous urban driving. In *Autonomous robots* (p. 123-129).

Schreier, M., Willert, V., & Adamy, J. (2013). From grid maps to parametric free space maps - a highly compact, generic environment representation for adas. In

Intelligent vehicles symposium (iv), 2013 ieee (pp. 938–944).

Sivaraman, S., & Trivedi, M. M. (2014). Dynamic probabilistic drivability maps for lane change and merge driver assistance. *IEEE Transactions on Intelligent Transportation Systems*, *15*(5), 2063–2073.

Thomas, J., Stiens, K., Rauch, S., & Rojas, R. (2015). Grid-based online road model estimation for advanced driver assistance systems. In *Ieee intell. veh. symp. proc.* (Vol. 2015-Augus, pp. 71–76). Institute of Electrical and Electronics Engineers Inc.

Triebel, R., Pfaff, P., & Burgard, W. (2006). Multi-level surface maps for outdoor terrain mapping and loop closing. In *Intelligent robots and systems, 2006 ieee/rsj international conference on* (pp. 2276–2282).

Urmson, C., Anhalt, J., Bagnell, D., ..., Darms, M., & Ferguson, D. (2008). Autonomous driving in urban environments: Boss and the Urban Challenge. *J. F. Robot.*, *25*(8), 425–466.

Wang, Y., Shen, D., & Teoh, E. K. (2000). Lane detection using spline model. *Pattern Recognit. Lett.*, *21*, 677–689.

Weigel, H., Cramer, H., Wanielik, G., Polychronopoulos, A., & Saroldi, A. (2006). Accurate road geometry estimation for a safe speed application. In *2006 ieee intelligent vehicles symposium* (p. 516-521).

Weiherer, T., Bouzouraa, S., & Hofmann, U. (2013). An interval based representation of occupancy information for driver assistance systems. In *Intelligent transportation systems-(itsc), 2013 16th international ieee conference on* (pp. 21–27).

Weiss, T., Schiele, B., & Dietmayer, K. (2007). Robust Driving Path Detection in Urban and Highway Scenarios Using a Laser Scanner and Online Occupancy Grids. In *Ieee intell. veh. symp. (iv 2007)* (pp. 184–189). Istanbul, Turkey: IEEE.

Weisswange, T. H., Bolder, B., Fritsch, J., Hasler, S., & Goerick, C. (2013). An integrated ADAS for assessing risky situations in urban driving. In *Intell. veh. symp. (iv), 2013 ieee* (pp. 292–297).

Ziegler, J., Bender, P., Dang, T., & Stiller, C. (2014). Trajectory planning for Bertha - A local, continuous method. In *Ieee intell. veh. symp. proc.* (pp. 450–457). Institute of Electrical and Electronics Engineers Inc.