UNIVERSITÀ DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXX Ciclo

AUTONOMOUS NAVIGATION, FROM SENSING TO PLANNING

Coordinatore:

Chiar.mo Prof. Marco Locatelli

Tutor:

Chiar.mo Prof. Pietro Cerri

Dottorando: *Daniele Molinari*

Anni 2014/2017

A tutti coloro che trovano quiete solo nel miglioramento,
gli eterni insoddisfatti,
gli instancabili camminatori.

List of Contents

List of Figures

List of Tables

Introduction

This thesis presents the research path I followed during my three years of PhD at the Università degli Studi di Parma, working in the VisLab research laboratory.

During these years I worked on different technologies that enable the possibility of autonomous driving. I contributed to the development of different levels of autonomy on three robotic car platforms, facing both the theoretical and practical development challenges of this research.

The focus of my work is *autonomous navigation* and covers the problem of detecting the navigation reference from sensors and actually driving the car on that reference in compliance with the road rules. The results of this work have been validated on VisLab's latest autonomous vehicle.

The following introduction frames the problem of autonomous driving into a context that sustain the decisions made about the many-choices faced when approaching the problem of autonomous driving; these choices had a profound impact and delineated the development of my research.

0.1 Why autonomous driving

Autonomous driving, AD for short, refers to the ability of a particular vehicle, usually called *autonomous vehicle* (AV), to carry out driving tasks with potentially limited human supervision.

This maturing technology applied to personal mobility has the potential to

reshape the way society approaches the transportation problem. Motivated by the opportunities conveyed by AD, the last decade has seen a quickly increasing research efforts, in both the academia and industry, toward reaching a reliable level of automation. The potential benefits of AD impact multiple aspects: safety, financial, health and quality of living.

Maybe the most obvious motivation for AD is the goal of reducing road fatalities. As reported by CARE[1] statistics, in the year 2015 the total number of road fatalities was 26100 in the EU [1]. In a similar fashion, the latest NHTSA[2] report indicates a total of 35092 fatalities in the US [2]. Of these casualties, about 94% are attributed to human errors with 31% involving intoxicated drivers and 10% due to driver's distraction [3]. AV would certainly be able to eliminate the risk factor associated with an altered state of the driver (distracted, sleeping, intoxicated, etc...) or with deliberate violation of the road rules. This potential safety gain is confirmed also by the latest NHTSA strategic plan that included deployment of automated vehicle as a strategic goal [4]. The potential reduction of accidents related to the driver's performances is still unclear, as some study suggests [5].

Another potential benefit of AV is the ability to reduce traffic congestion, [6] shows that even a single AV can dump the effect of the so-called *phantom jams*. This would impact fuel consumption, environmental pollution and would help reduce the harming effect of driving stress in traffic jams [7].

A further major impact of AV would be the recover of the actual driving time. Estimates suggest that, in US, the average citizen spend 17600 minutes driving every year [8, 9]. The redeem of this time, converted to productivity or leisure, would have an estimated financial impact of around $1.2T a year, bigger than all the previous factors considered together [10].

Finally, it is well known that car are used less than 10% of their life time [11]. Based on this consideration, a new paradigm of personal mobility is pos-

[1]The European Union's road accidents database `https://ec.europa.eu/transport/road_safety/specialist/statistics_en`

[2]National Highway Traffic Safety Administration `https://www.nhtsa.gov/`

sible: combining car sharing with AD to provide transportation on demand. A case study in Singapore suggests that this paradigm is a financially convenient alternative to the classic transportation model reducing the number of vehicle up to 1/3 [12].

0.2 Brief history of autonomous driving

The idea of a "driverless" car dates back to the 20's. It was 1925 when Houdina Radio Control Co. equipped a car with radio communication and publicly demonstrated a drive in New York's street with no one on board [13]. Named *American Wonder* (figure 1), the car had motors to control the movement and was teleoperated by a following car. The car was demonstrated again in 1932

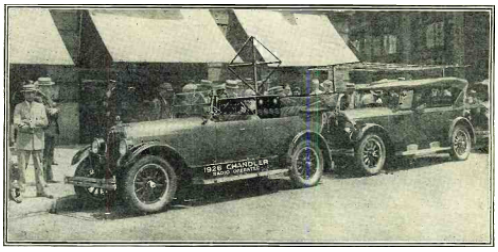


Figure 1: American Wonder car.

in Fredericksburg with the name of *Phantom Auto*, the local paper "The Free Lance-Star" was writing:

> [...] "Phantom Auto" will be piloted through the streets of the city without a driver or occupant, with no wires or strings attached to it.
>
> It sounds unbelievable but it is true [...] [14]

While not being properly autonomous, since full human supervision was required, this was probably the first step toward AV.

In 1939, at the New York World's Fair, Norman Bel Geddes demonstrated the first car with automated driving. The car was driving thanks to electronic circuits embedded in the pavement. In his book Magic Motorways, Geddes also advocated the removal of humans from the task of driving [15].

While in 1956 America's Power Companies' were advertising a future with autonomous vehicles, as shown in figure 2, the research continued in the direction of vehicle driven by a road infrastructure equipped with proper circuit in the pavement. While these products were expected to become a reality by

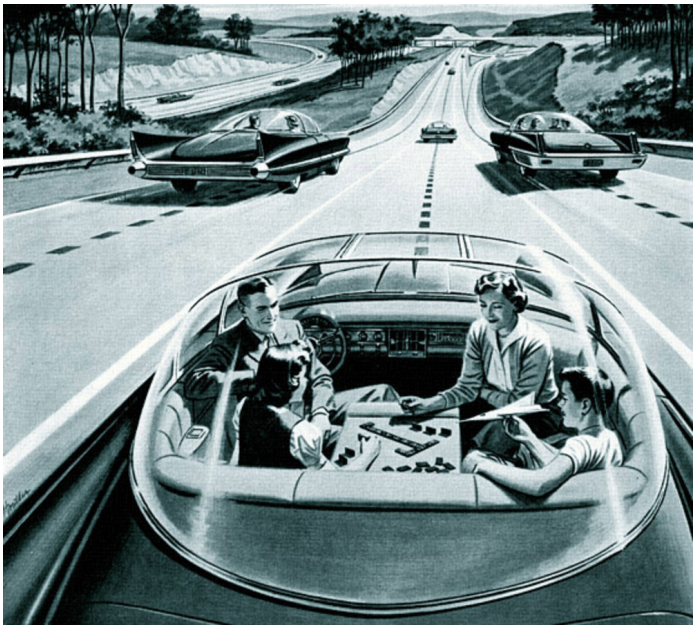


Figure 2: America's Power Companies' advertisement showing a future with autonomous vehicles driven by a smart road infrastructure.

the mid 70's, the funding of this research was later dismissed.

Later in '80s the pioneering work led by Ernst Dickmanns made AV a real possibility. With his team at the Bundeswehr University of Munich they drove a Mercedes-Benz van up to a speed of 63km/h on streets without traffic [16].

In the same years the massive research project PROMETHEUS (€749M)

was funded to develop autonomous vehicles. The culminating demonstration was in 1994, when the VaMP AV drove over 1000Km with top speed of 130Km/h in a multi-lane highway in Paris [17], figure 3.

Figure 3: The vehicle VaMP autonomously driving in a multi-lane highway.

In 1989 the Carnegie Mellon University pioneered the use of neural network to drive the car directly from camera images [18], figure 4.

In 1995, the CMU Navlab demonstrated further progress with 2849 miles driven with automated steering across America [19].

1996 was the year professor Alberto Broggi, from the Università degli Studi di Parma, with the project ARGO lunched the "Mille Miglia in Automatico" (*one thousand automatic miles*) traveling 1900Km over six days in the highways of north Italy. The modified Lancia Thema (figure 5) was using the lane marks to drive in the unmodified highway environment, detecting obstacles and other vehicles. About 94% of the journey was in automatic mode with the longest stretch being 55Km [20].

The years 2004-2007 witnessed a big leap forward thanks to the DARPA funded Grand Challenge competitions. The first challenge was in 2004 and required the participant to finish a 150 miles drive in the Mojave desert without any human intervention [21]. The $1M prize remained unclaimed as no team

Figure 4: ALVINN autonomous vehicle, driven by a neural network.

were able to complete the course. Figure 6 shows example of failures experienced by the teams. In 2005 the challenge was repeated and 5 teams were able to complete the race [22]. DARPA later founded the Urban Challenge in 2007, vehicles were required to cooperate in a fully simulated urban environment with traffic, pedestrian and GPS denial [23], figure 7.

In 2010 the VisLab team, led by professor Alberto Broggi, ran the VisLab Intercontinental Autonomous Challenge (VIAC): a 15900Km drive from Rome to Shanghai. This marked the first intercontinental land journey completed by autonomous vehicles [24] (figure 8). Again in 2013, VisLab completed the PROUD (Public ROad Urban Driverless) test: the vehicle BRAiVE successfully drove from the university campus to the downtown of Parma without human intervention. The 13Km drive comprised different road types (rural, urban and highway) and the car had to negotiate junctions, roundabout and traffic lights [25, 26]. Picture 9 shows the arrival in the downtown.

Later the same year, Daimler R&D made a Mercedes-Benz S-class drive autonomously the historic Bertha Benz Memorial Route, about 100Km from Mannheim to Pforzheim, Germany [27].

In the following years the number of research projects and demonstrations

Figure 5: The ARGO car driven about 1900Km of highway with automated steering.

grew exponentially. Notably in 2014 Google unveils some prototypes of AV and the research project, secretly started in 2009, continues as Waymo in 2016 [28], figure 10. In 2015 Tesla Motors introduces the Autopilot capability, enabling automation of steering and gas/brake in favorable conditions [29]. 2016 sees the introduction of AV fleets in Pittsburgh by UBER and in Singapore by nuTonomy, two new startups born out of this increasing interest toward AD.

It is now evident that the financial interest in this technology, and the consequent engagement of the industry, turned the development of AD into a real race. These years are exciting times for the AD research and the final

Figure 6: Example of failure during the desert ride.

goal seems, like never before, really at hand.

Figure 7: During DARPA Urban Challenge vehicle were operating in human traffic. The other vehicles were driven by stunts obliged to follow the road rules.

Figure 8: Porter: the electric vehicles that faced VIAC, equipped with solar panel for extended autonomy.

Figure 9: The BRAiVE car negotiated regular urban traffic finishing its drive in the very downtown of Parma. The final stretch was driven with no person on the driving seat.

Figure 10: Google's self driving car.

Chapter 1

Autonomous Driving: Problem Statement

This chapter contains introductory material to the problem of autonomous driving (AD) and the platforms used to develop the relative solutions: autonomous vehicles (AV).

First, the official classification autonomy's degree is introduced, followed by an overview of the typical AV functionalities and hardware. Lastly, the main approaches to the AD problem are presented, highlighting the choices made in the context of this research and the impact they had.

1.1 The different levels of autonomy

Over the years, the term *autonomous vehicle* has been used in a somewhat ambiguous way and the previously given definition ("[...] carry out driving tasks with potentially limited human supervision") is intentionally vague to account for the many shades of meaning attributed to it.

To set a common ground when talking about AD, in 2014 SAE International published the J3016 standard, named "Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems", that

establish 6 automation level, from 0 (full manual) to 5 (full autonomous) [30]. The latest 2016 revision has been adopted also by NHTSA to uniform the definitions regarding AD.

To understand the different levels, it is important to introduce some definition. The driving task is divided in 3 aspects:

- *operational*: operation of steering, throttle/brake and monitoring of the vehicle and roadway.
- *tactical*: responding to external events, determining when to change lane, turn, use signals, etc.
- *strategic*: determining destinations. In the following it is assumed that this aspect always falls on the human driver/passenger.

Moreover the term *driving mode* refers to a specific driving scenario like: highway, high speed cruising, low speed traffic jam, etc.

Each autonomy level is detailed in the following:

Level 0 – No Automation : the driving task falls completely on the human driver, even when aided by warning or intervention systems. This includes car with no automation of either steering or throttle/brake, even when equipped with systems like *ABS*, *lane departure warning*, etc. . . .

Level 1 – Driver Assistance : mode-specific automation of either steering or throttle/brake. The human driver is expected to monitor the environment and perform the remaining driving tasks. In this category fall the cars with *lane keeping* and *adaptive cruise control* functionalities.

Level 2 – Partial Automation : mode-specific automation of both steering and throttle/brake. The driver is expected to monitor the environment and perform the tactical aspect of driving.

Level 3 – Conditional Automation : mode-specific automation of the operational and, possibly, strategic aspect of driving. The driver is expected to take over the driving when warned with a “request to intervene”.

Level 4 – High Automation : mode-specific automation of the operational and strategic aspect of driving. The system is expected to behave properly even when the driver doesn't respond to a "request to intervene".

Level 5 – Full Automation : full automation of every aspect of driving, under every scenario and condition that can be managed by a human driver.

The important aspects of each level are summarized in table 1.1.

Level	Operational Autonomy	Environment Monitoring	Mode
0	none	human	n/a
1	partial	human	restricted
2	full	human	restricted
3	full	system+human fallback	restricted
4	full	system	restricted
5	full	system	everywhere

Table 1.1: Summary of the SAE automation levels.

Of course these categories are not perfectly separated and there can be applications that are hybrids of different levels.

1.2 Introduction to autonomous vehicles

In the context of this thesis, Autonomous Vehicle (AV for short) is the term used to indicate any robotic-car platform used to develop/deploy autonomous driving. The terms *agent* and *robot* will also be used when referring to AV.

We can think of the AV as a robotic agent immersed into a dynamic environment it needs to interact with. Like any agent of this sort, the AV need to perform the following operations:

- perceive the surrounding environment.

- understand the state of the relevant elements.
- act accordingly.

These operations define the three main functional area for AD: *perception*, *data fusion* and *planning*. As indicated in figure 1.1, the main data flow goes

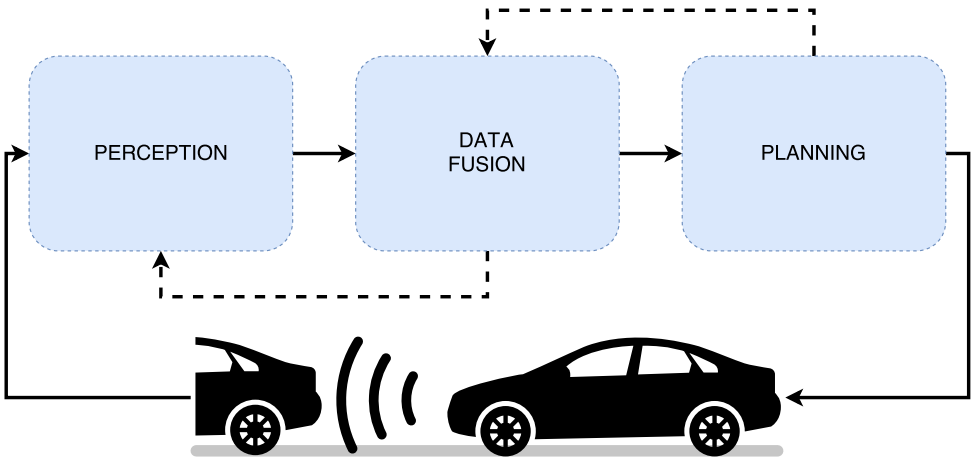


Figure 1.1: AV main functions: perception, data fusion and planning.

from perception to data fusion and, from there, to planning (solid arrows). It's anyway appropriate to include also feedback connections (dashed arrows) because the data fusion might switch focus given the future actions and the perception might use informations coming from the data fusion, like the refined vehicle attitude.

To perform these functions the AV also requires some additional hardware. The functionalities and hardware are the focus of the next sections.

1.2.1 Perception

Any physical agent interacting in a dynamic environment needs some form of perception. When the state of the surrounding is not completely known *a priori*, the robot need to directly sense it to infer its actual state. Except

for completely controlled experiment, this is the case for AV because its environment has some dynamic characteristic. While we can assume to know the position of the road in advance (for example with a pre-recorded map), there are elements that are intrinsically dynamic. First among all, there are other road users: their state (position, speed, etc.) can't be predicted without some direct perception and they greatly affect the required driving behavior. Secondly there are road infrastructures that change state over time, like traffic light and rail crossing. Finally, to achieve the higher levels of autonomy, the agent must properly react to unexpected conditions, being that road work, traffic policemen or roadblocks. Moreover, since maps are not always viable, also the perception of the road geometry and topology is of great interest, along with street sign and other road features.

1.2.2 Data fusion

Data fusion refers to the process of aggregating data coming from different sources to extrapolate a richer representation. This operation, also called *scene understanding*, has the general goal of providing an estimate of the most-likely state of the world. This definition is quite general because the range of tasks performed can be broad depending on the actual overall architecture.

A typical data fusion task is obstacle tracking. This amount to using multiple detections, possibly coming in different forms, to provide a stable and accurate estimate of the obstacles' state. When information about the road topology/geometry is available, other vehicle's state can be augmented with high level information like lane of travel and future action estimate (lane change, turn, etc.).

Another example of data fusion task is ego localization: providing an accurate estimate of the AV position w.r.t. some reference system. At the basic level this means fusing IMU information with the GPS signal. When rich map are available, the localization can be refined comparing the detected features with the recorded one.

1.2.3 Planning

To achieve AD, the vehicle need to act according to the world state: the decision process is called *planning*, while the actual action is delegated to the *control* system.

The planning system is responsible for determining an appropriate behavior for the car and finding a reference trajectory that implements that behavior.

Examples of behavior might be: follow the current lane, change to the left/right lane, yield at the intersection, etc. This high level description must be translated to a reference trajectory: a series of future positions in the space and velocity set-points that the vehicle should follow.

The task of having the car actually following this reference is a problem by itself and it's covered by the *control theory*. Basically the vehicle is a dynamic physical system and imposing positions/velocities in its configuration space require some form of feedback.

One remark is needed about the proposed division between perception, data fusion and planning. This structure is inherited by the robotic field and provide a clean division between tasks [31]. Some sources omits the data fusion part, collapsing it into the perception system [32, 33]. I advocate this distinction because detection and fusion operate at different abstraction and semantic level, using a set of entirely different techniques. This architecture has the advantage of providing a system made of *explainable* sub-parts, that can be developed almost independently, but it's not the only possible one. In particular, with the recent advancement in machine learning a new paradigm has emerged: end-to-end driving. This paradigm is based on a single neural network that computes driving command directly from camera images [34, 18]. Although this represent a significant advance in the field, such a system is not suited for real applications because it offers little insight on its internal behavior, limiting the possibility of tuning and fixing undesired results.

1.3 Autonomous vehicle hardware

Cars are historically designed to be driven by humans. To support AD, some form of specialized hardware is needed; this can be divided in three different categories: *sensors*, *computing hardware* and *drive-by-wire*.

1.3.1 Sensors

We can divide sensors in two categories: *proprioceptive*, that give information about the ego vehicle, and *exteroceptive*, that sense the external environment.

Exteroceptive sensors are the primary mean for the vehicle to perceive the world around. They feed the input to the perception system in forms of images, scans or echoes.

Cameras observe the surrounding capturing images that can be processed. Images are a very rich representation of the world, but this dense information is typically hard and expensive to process. They need an appropriate quantity of light to work properly, so they're sensitive to scarcity (e.g. night, tunnels, etc.) or excess (e.g. sun blinding) of light.

LIDAR[1], also called laser scanner, measure the distance of objects using a revolving laser beam. They provide scans containing the distance given the horizontal and vertical angle of the beam. The information is poor in semantic, but the distance are measured with great precision and accuracy. Being active sensors, they're also relatively insensitive to light condition.

RADAR[2] emit electro-magnetic pulses and, through the echoes, measure position and speed of reflecting objects. Their information is sparse and very straightforward to process[3]. They're analogous to LIDARs but work in a much lower band-width that gives them different characteristics. They have extremely high range of detection, but also a sensible minimum distance. They're

[1] LIght Detection And Ranging

[2] RAdio Detection And Ranging

[3] This claim comes from the observation that RADAR doesn't output raw data, but the outcome of some processing.

not as accurate as LIDARs, but their ability to measure speed is valuable.

SONAR[4] measure distance of close object emitting acoustic pulses. Their range is extremely short and the information is far from being precise, but for application like park assist/autonomous parking they're still useful.

Proprioceptive sensors are used to measure the state of the ego vehicle. They usually provide informations relative to the position and movement of the vehicle that are used to perform proper localization.

The GPS[5] is the primary mean to have an absolute localization. The localization precision it can supply depends on multiple factor: the number of satellites, the power of the signal and the availability of augmentation services. When corrected with augmentation services it's usually called GNSS, which stands for *Global Navigation Satellite System*, to emphasize that the reached precision is sufficient for vehicle navigation. Even with the most advanced platforms, the urban scenario poses the hard challenge of severe GPS outage. In the urban canyons, reflection and multiple paths for the signal can fool the position around leaving little clue that something wrong is happening (the number of satellites and the signal power may look good).

IMU[6] data are often used to refine GPS position. These platform provide data about the acceleration and angular rate of the vehicle, information that can be used to locally smooth the GPS movement.

Finally the car itself usually provide some odometry information like speed coming from the wheel encoders, internal IMU, engine power, braking torque, etc.

1.3.2 Computing hardware

For the functions of perception, data fusion and planning to take place, the car need some form of computing hardware. Usually one or more PC are present on board, but the actual hardware really change from vehicle to vehicle. For

[4] SOund NAvigation and Ranging
[5] Global Positioning System
[6] Inertial Measurement Unit

example, in the case of [34] the car is equipped also with powerful GPU's to support the neural network inference.

For the future of AV, it's commonly agreed that the computing will be performed on some specialized hardware, reducing encumbrance and power consumption.

1.3.3 Drive-by-wire

The driving devices present on cars, steering wheel and throttle/brake, are designed to be operated by humans. To let a computer drive the car, some form of interface is necessary: the drive-by-wire.

On car that are fully manual, specialized actuators are installed to mechanically operate the steering wheel and the gas/brake pedals. These electric engine can be controlled via some communication interface (typically CAN, but might be a different technology too), setting appropriate set-points.

Some recent car models are designed to natively support operation of steering and gas/brake via electronic signals. These systems are currently used to implement ADAS and, with little modification, are suitable also for AD.

1.3.4 Communication hardware

Some branches of development go in the direction of implementing a thick communication network connecting the vehicles and the road infrastructure [35, 36]. In this paradigm, called V2X, the vehicles will communicate with each other transmitting the current state and the future intentions. In the same fashion, the road infrastructure will play an active role, arbitrating the negotiation of junctions and providing the vehicles useful information such as the position of stop lines, state of traffic-lights, etc...

It can be assumed that this will improve the environmental awareness of the vehicles, but as long as the road remain accessible to 'entities' not connected to this network (being those pedestrians or regular vehicles), the necessity for a complete and reliable perception system persists.

1.4 Ways to autonomous driving

As previously mentioned, there are many ways to tackle AD depending on the context in which the problem is framed. Each context imposes different hardware choices and limitations that greatly affect the requirements and development of the perception, data fusion and planning systems.

Following [10], the AD applications are analyzed considering separately the level of automation (< 5) and the scale and scope of application, as shown in figure 1.2. The top-right corned of this diagram corresponds to the NHTSA

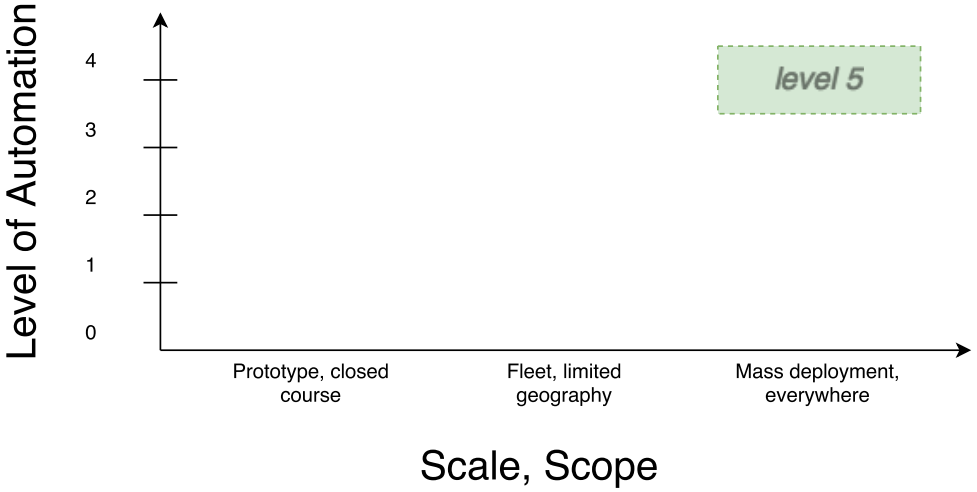


Figure 1.2: Analysis of AD applications separating the automation from the scale and scope of application.

level 5 (full autonomy everywhere) and it's presumably the ultimate goal of AD research.

The next two sections presents the two major contrasting path to level 5: the one followed by startups and the one adopted by carmakers/OEM. Their approach is antithetic because the scope of the problem is entirely different. Ultimately the approach followed by VisLab is presented, an hybrid that blends elements from the other two.

1.4.1 Startup's way

The newborn startups, sometimes generated from the academia, are the leading edge in terms of achieved automation. Examples of such startups are Waymo [28], nuTonomy [37] and Uber ATG [38].

Their common feature is that they see the AV as a service as opposed to a product. This vision enables the mobility-on-demand paradigm, capturing the benefits of car sharing. From the financial perspective it also justifies the use of expensive hardware since the vehicle is company owned. Moreover, the service can be deployed in limited areas and suspended in case of adverse condition, limiting the requirements for the initial deployment.

Since full autonomy (level ≥ 4) is essential to pursue this way, the startups take advantage of the advanced hardware they can use and their effort is directed toward expanding the scope of level 4 platforms.

Their research path is thus summarized in figure 1.3.

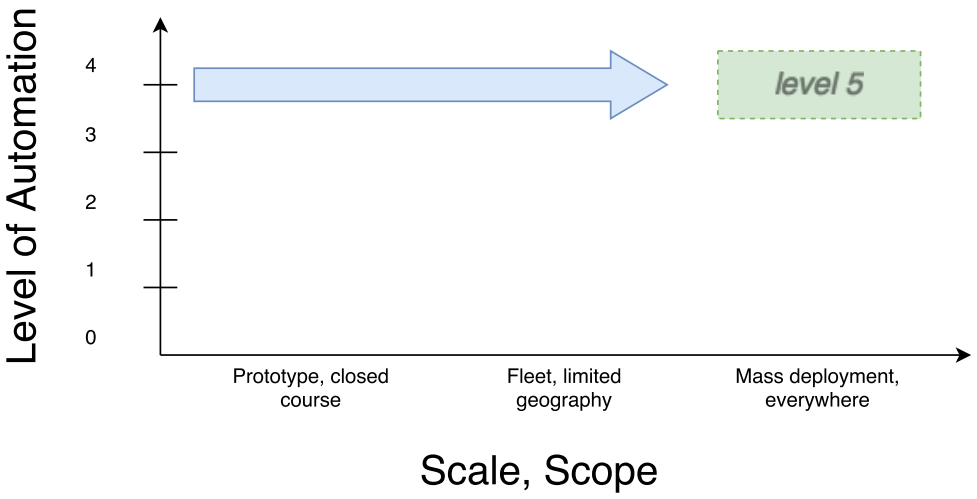


Figure 1.3: Startup way to AD. They work toward expanding the scope of full autonomous platforms.

1.4.2 Carmaker & OEM's way

On the opposite side, carmakers and OEM[7] see the AV as a product. This could mean actually having driverless-car for sale or selling AD as an optional feature on selected car models. The biggest constraint this vision imposes is the actual price of the final product: keeping this in a reasonable range requires the use of low-cost components. Furthermore, the path to level 5 must pass through *sellable* intermediate products.

For these reasons, the research effort from these companies is directed toward improving the autonomy on platforms that can be readily mass-deployed, as illustrated in figure 1.4.

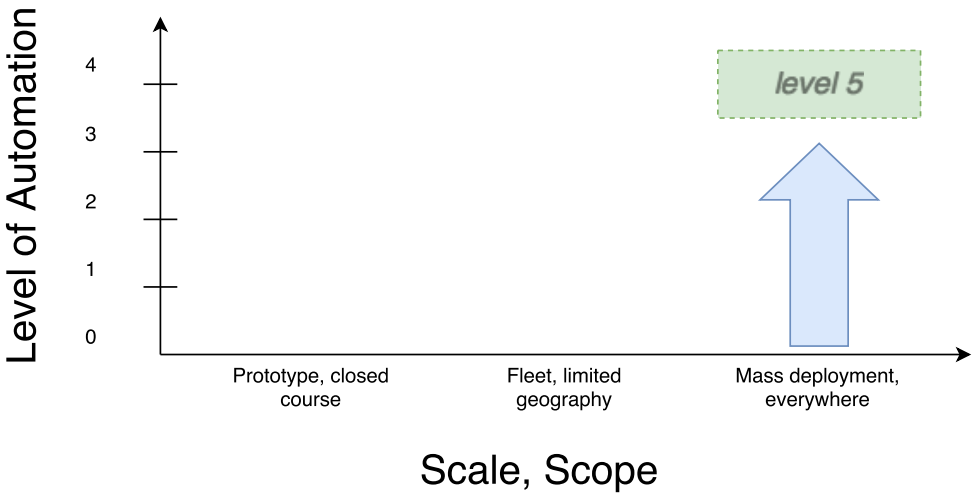


Figure 1.4: OEM and carmakers way to AD. They work toward improving the autonomy of mass-deployable platforms.

Achieving high level of automation using low-cost components requires to solve additional challenges and is inherently harder compared to the case with no cost limitations.

[7]Original Equipment Manufacturer, the original producers of vehicles' components.

1.4.3 VisLab's way

Both the presented ways are viable research paths to AD, each justified by the relative problem context. They represent quite opposite approaches and intermediate solutions are also valid. An example is the approach adopted in our research laboratory at VisLab. Born as a spin-off from the Università degli Studi di Parma, VisLab has always been in contact with major players in the automotive field. With this target in mind, the goal of the research has been to develop high automation applications using technologies that result appealing in the automotive context.

Summarized in figure 1.5, the research path worked in both the directions of improving the system automation and widening the range of handled scenario, always using low-cost solutions and automotive hardware.

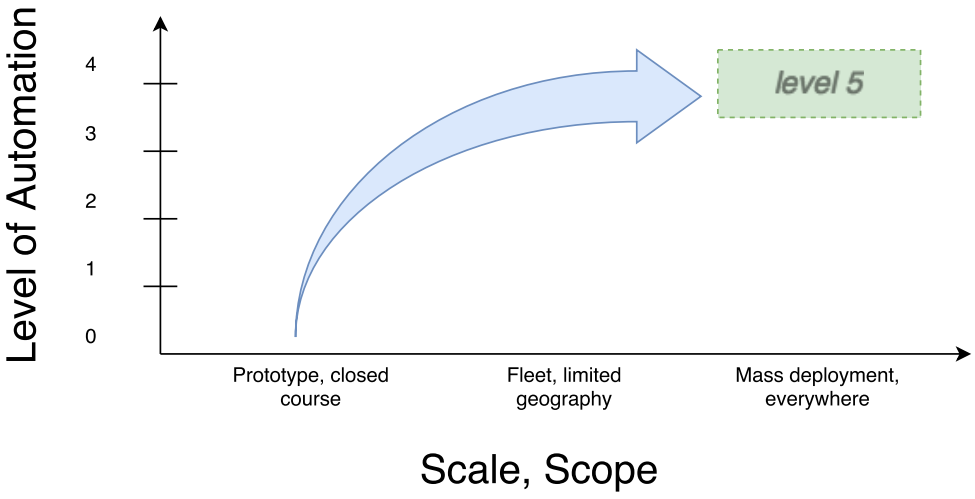


Figure 1.5: VisLab adopted an intermediate solution toward AD, delivering high automation using only automotive hardware.

As already mentioned, using low cost/automotive hardware poses additional problems to be solved. The next sections will highlight the major challenges and the way they affected the development of my research.

Allowed sensors

A big restriction concern the actual sensors the vehicle can mount.

The precise and reliable LIDAR is not suited for automotive application because of the excessive cost and the high power consumption. For this reason, the preferred sensors are cameras, RADAR and SONAR. These sensors already made their way into the automotive field and conform the requirements in terms of price and power consumption.

As a result, the precise 3D measurements of LIDAR are not available. This information can be recovered sparsely using RADAR or in a dense manner with stereo-camera technology. The major impact of this limitation regards obstacle detection.

Localization and maps

A typical solution to the vehicle localization problem is to use a precise INS that combines GNSS and IMU (e.g. PINPOINT[8] and OxTS[9]). Those platforms provide an accurate and smooth absolute localization, but are typically very expensive and thus not viable in our scope. Another possible solution, is to infer the vehicle position confronting the detected features with a database of localized features mapped in advance. While it's hard by itself to guarantee a high precision due to detection noise, it's also not safe to assume the availability of this augmented maps everywhere.

For these reason we generally don't want to rely on precise global localization. The absence of global positioning, also negate the possibility of using mapped road paths as a navigation reference. To have a consistent driving it's therefore necessary for the AV to perceive the road path using the onboard sensors. This problem motivates my work in road-path detection and estimation and will be covered in more details in the following chapters.

[8] http://torc.ai/pinpoint/
[9] http://www.oxts.com/

Computation power

The cost and power consumption limits cast severe limitations also on the available computing power. While the prototype vehicle can be equipped with regular PC, extra caution must be taken during algorithms development in perspective of moving to low-power platforms.

For this reason, computational efficiency will be a major concern throughout this thesis. This limitation had a big impact especially in the development of the planning system, a function that's notoriously computation hungry.

1.5 Test vehicle setup: DEEVA

The work presented in this thesis has been implemented, tested and validated on an actual autonomous vehicle: DEEVA [39].

DEEVA,figure 1.6, is the latest AV platform developed at VisLab and represents a significant effort toward *automotive-compatible* robotic cars. Based

Figure 1.6: The latest VisLab's autonomous vehicle: DEEVA.

on a AUDI A4, this car is equipped with sensors,computers and drive-by-wire to support AD development.

The major distinctive feature of this platform is that all the sensors and additional hardware have been seamlessly integrated and, as the picture shows, the overall look is one of a regular car. The next sections describe the hardware installed in this platform.

1.5.1 Sensors

DEEVA, being a prototype, has a large number of sensors that provide total coverage of the environment, redundancy and act as a ground truth during the development process.

A peculiar trait is the presence of 26 cameras, organized in 13 stereo couple placed in the positions indicated by thew gray boxes in figure 1.7.

Image 1.8 shows the field of view (FOV) of the installed cameras, divided in near (1.8a) and far (1.8b).

Beyond cameras, DEEVA mounts 4 LIDAR that cover the front and the back of the vehicle, as indicated by the red boxes in figure 1.7. Their primary use is as ground truth for some perception algorithms, mainly obstacle detection.

A long-range RADAR is installed in the front of the vehicle, indicated by the green box in figure 1.7. This RADAR is used for long-range vehicle detection and high speed adaptive cruise control.

Images 1.9,1.10 and 1.11 show the final look of sensors integration.

Finally, the car install a OxTS RT-3300: an high precision INS platform that provides global positioning with an accuracy of few centimeters. This is used as a ground truth for the vehicle localization.

1.5.2 Computing hardware

To process the huge stream of data coming from the 26 cameras, the entire trunk is devoted to host the computational backbone of the car, figure 1.12.

Up to 20 industrial-grade PC are installed in four racks. The communication between computers and sensors is supported by 3 Gigabit Ethernet

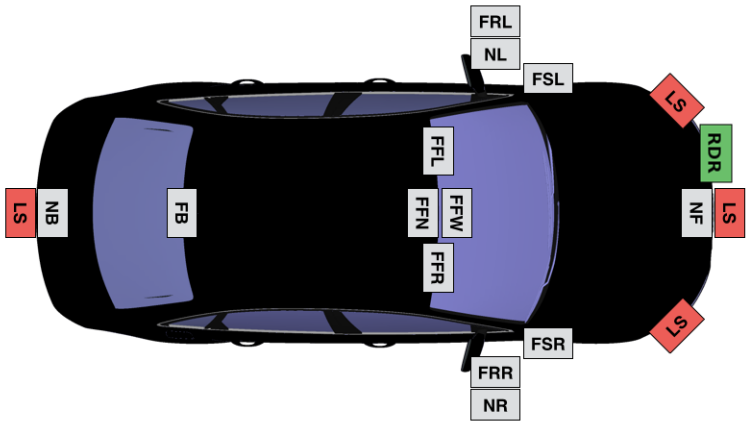


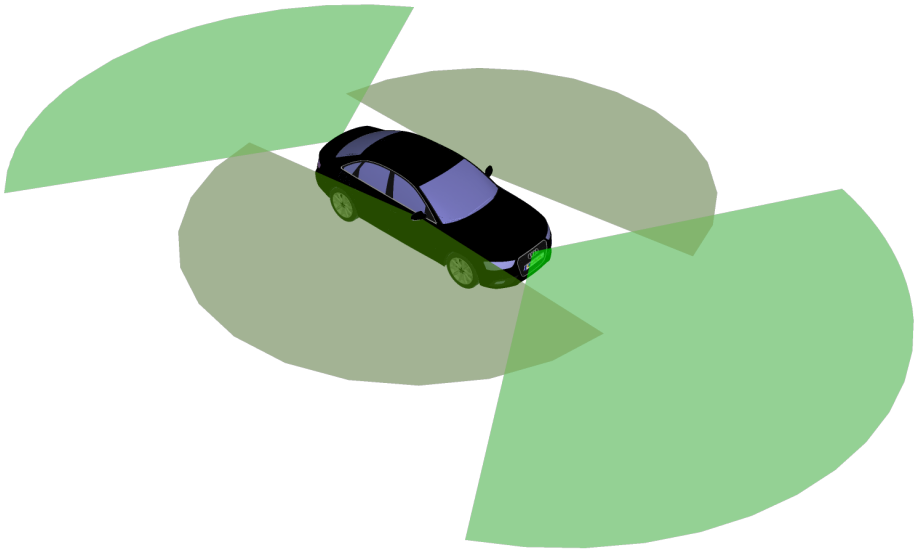
Figure 1.7: Sensor suite on DEEVA: 13 stereo couples (gray boxes), 4 laser scanners (red boxes) and radar (green box).

switches. This hardware architecture obviously impact also the software design that must distribute the computation among different nodes.

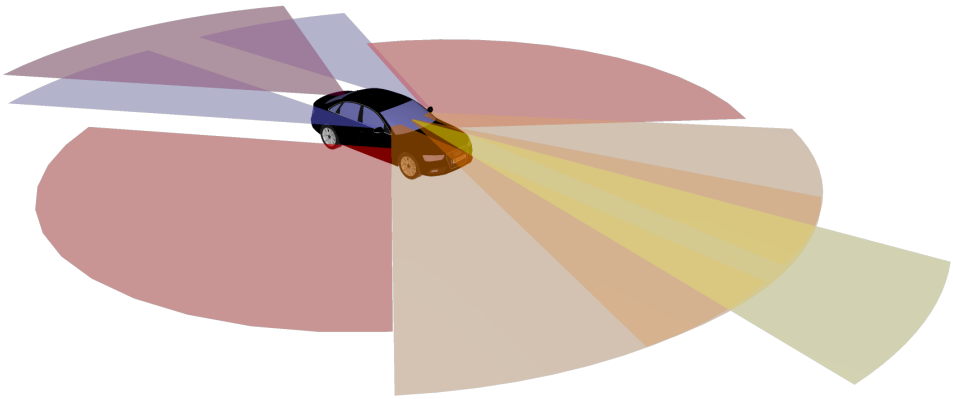
1.5.3 Drive by wire

DEEVA is equipped with the AEVIT RPV drive by wire system. The main components are the external mechanical actuators intended to operate the steering wheel and the gas/brake pedals.

Image 1.13 highlights the actuator installed in the steering column. The electric engine is integrated inside the column and the yellow ball marks the device to engage/disengage the mechanical coupling.



(a) Near cameras FOV



(b) Far cameras FOV

Figure 1.8: Field of view of cameras installed on DEEVA.

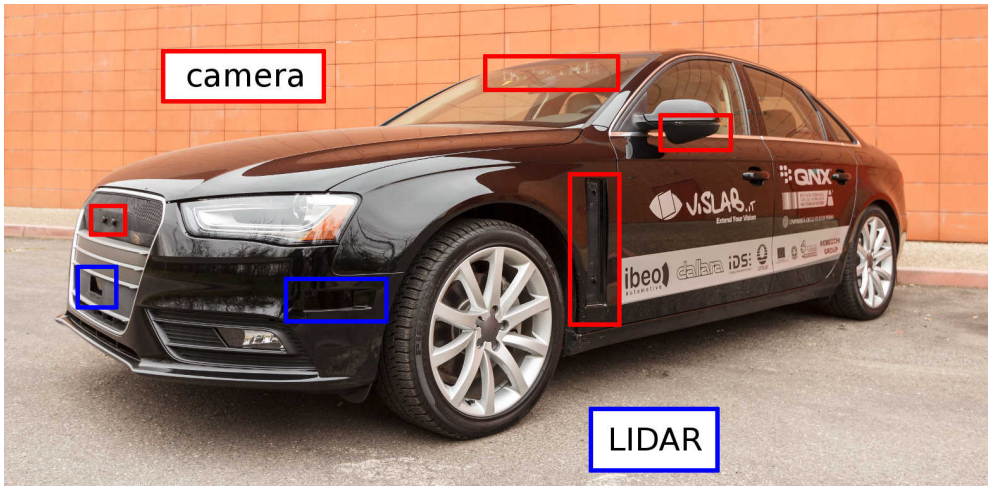


Figure 1.9: Final integration of cameras and LIDARs.

The gas and brake pedals are operated by another mechanical actuator, shown in image 1.14.

All actuators are fail-safe with redundant electric engine and can be operated via CAN interface. Set-points commands for steering and gas/brake are expected to be received at 50Hz, otherwise the system trigger a stop-state applying a configurable brake.

Also the gear shift is replaced by a control panel, shown in figure 1.15, that allows the gear to be controlled by both human drivers and computers.

Figure 1.10: The frontal camera rack. It provides medium and far range image coverage.

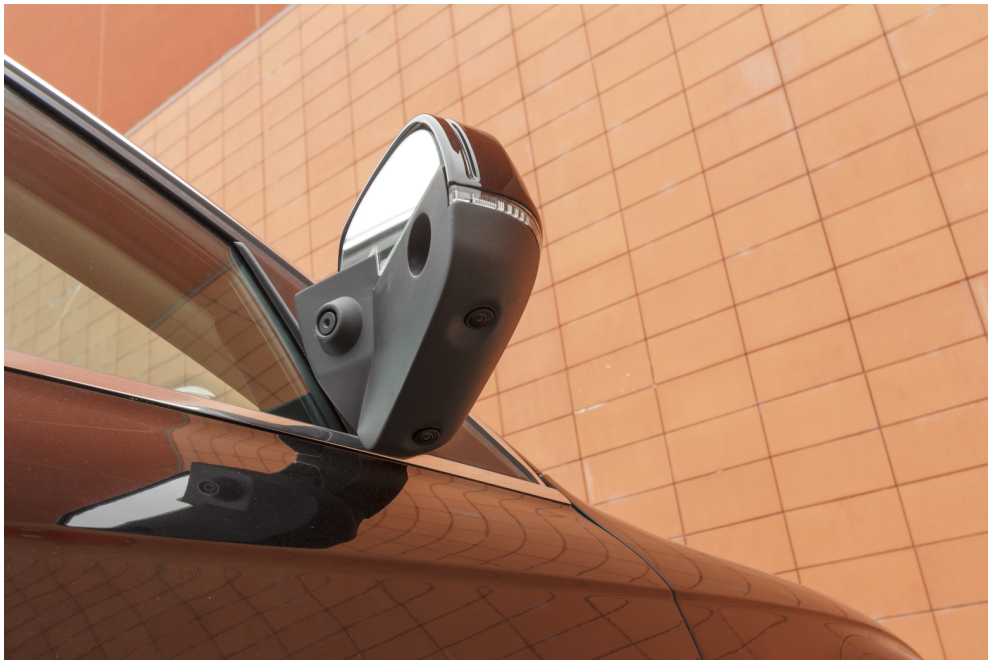


Figure 1.11: The mirror integrates 4 cameras: two fish-eye camera for lateral view and two long range, rear facing cameras.

Figure 1.12: DEEVA's computing hardware: up to 20 PC connected with 3 switches.

Figure 1.13: Steering wheel actuator.

Figure 1.14: Gas/brake pedals actuator.

Figure 1.15: AEVIT control panel.

1.6 Scope of work

This introduction delineated the context of my research.

First, solid motivations for the interest in *autonomous driving* are given, providing also an historic perspective on the research evolution.

Secondly, a general introduction to *autonomous vehicles* is presented, detailing the main functions such vehicles must execute and the hardware employed to support those functions.

Thirdly, the impact of the intended solution scope on the research path is outlined, highlighting the direction pursued by the VisLab research laboratory.

Lastly, the test vehicle DEEVA is introduced, being this the validating platform for the outcome of my research.

The focus of this work is autonomous navigation, meaning the detection of the road path from the sensory input, the stable estimation of the path geometric position and the planning process to drive the car on the reference. This problem spans over the three main function of autonomous vehicles and it's of paramount importance to enable autonomous driving. In figure 1.16 the components derived form this work are indicated in green.

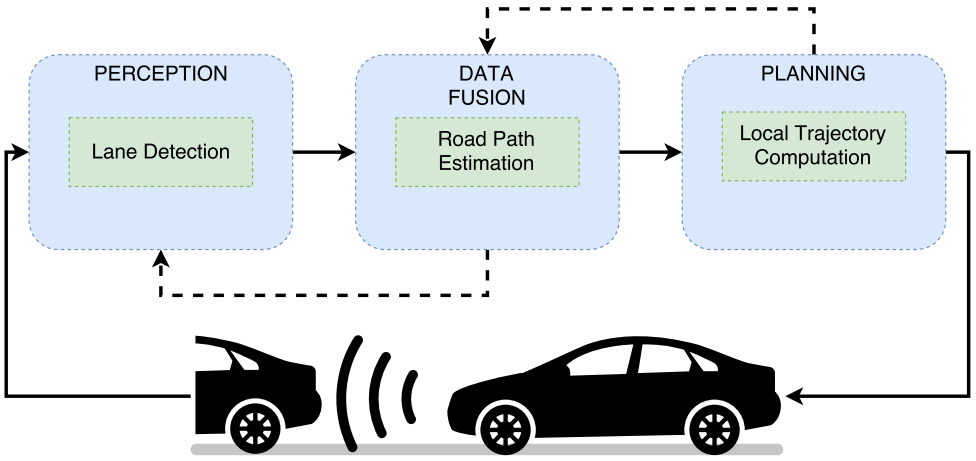


Figure 1.16: The components derived from this work are highlighted in green.

Chapter 2

Perception: Ego-Lane Detection

2.1 State of the art

Lane detection is an important component for both autonomous vehicles and advanced safety systems. A big advancement came with the DARPA Grand Challenge and Urban Challenge and, in the latest years, it received a lot of attention also from automotive companies since it is the key to ADAS[1] like *lane departure warning*[40] and *lane keeping assistance*[41]. While many approaches propose fusion of LIDAR and camera images [42, 43, 44], I restrict my attention to methods based solely on images for the reasons explained in the previous chapter.

In [45] the authors perform lane detection through lane markings detection and propose a two-steps method composed of a detection phase and a tracking stage. First, the camera image is transformed using the IPM[46] and the *dark-light-dark* patterns are highlighted [47]. The resulting points are clustered together and approximated with continuous piece-wise linear functions.

Secondly, the new poly-lines are confronted with the tracked lane markings

[1] Advanced Driving Assistance System

to produce new candidate and perform expansion operations to join pertinent non-connected components.

Finally the candidates are assigned a score and a threshold is used to produce the final result. Figure 2.1 shows some example of obtained results.

(a) Urban environment (b) Queued vehicles

Figure 2.1: Lane marking detection results.

A more recent approach for a *collision warning system* [48] integrates the disparity image to improve the results of lane markings detection. The processing is based on IPM, but the transformed image is thresholded removing the area relative to the obstacles detected using the disparity information.

This thresholded IPM is processed with a Sobel filter to highlight the edges of the lane markings. Hough transform is then applied to detect the dominant line and, from there, additional ROI are proposed in the Hough image to extract the other markings. Image 2.2 shows some results.

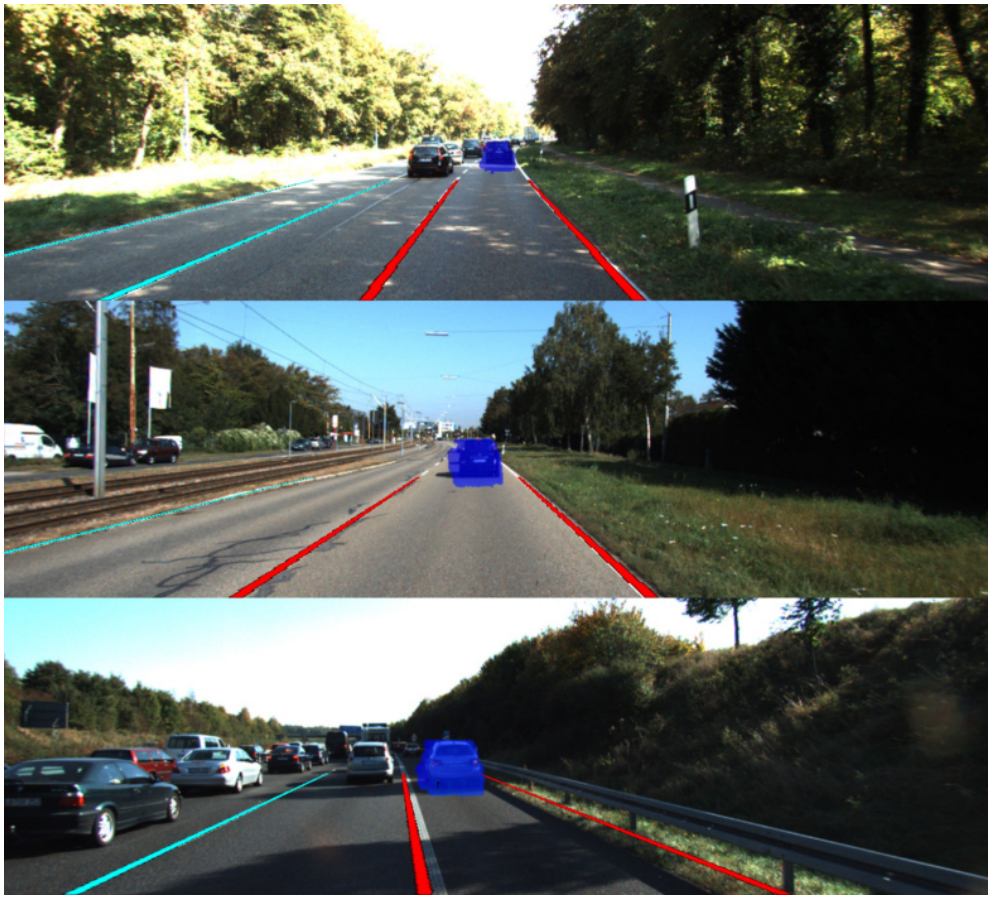


Figure 2.2: Final results for the collision warning system.

2.2 Limitations of classic lane detection

As can be deduced from the state of the art, the basic approach for lane detection in images hasn't changed much since 2010 and it's based on lane marking detection. This exposes a fundamental limitation: the approaches are constrained by the actual presence of the lane markings in the image. There are however many practical cases of interest in which the markings are poorly visible or missing altogether. Newly paved or old roads are a good examples of common situation with missing road markings (image 2.3). Image 2.4 shows an

Figure 2.3: Missing lane markings on a old road.

example of extreme shadow contrast that blacken the left marking. Reflections can be another source of problems, like in picture 2.5. Note, however, that in this case the image is not completely white and still contains the information about the course of the road.

Even when the markings are present, there is another problem in these approaches: the total lack of semantic in the detection process. Not every dark-light-dark pattern, or straight white line, in the image is a marking. In image 2.6 there is an example of misdetection due to the fact that the top of the curb offers a dark-light-dark pattern much stronger than the one of the

Figure 2.4: Left marking blacked by the guardrail shadow.

real marking.

For humans, the task of detecting those markings is trivial because we're familiar with the road environment and we put a lot of prior knowledge and semantic in the process. While the tracking phase and some hand-designed rules try to tackle this problem, they are somehow heuristic and can fail to achieve the intended result.

A further limitation is due to the fact that, like in [48], many approaches constrain the markings to fit a particular continuous model. Beyond straight line, which is very restrictive, other popular models are polynomials, clothoids and splines. The model choice is always a compromise between expressive power and noise rejection. The problem is that very restrictive models can fail to represent some topologies of interest, while models that are too expressive often surrender to the noise in the detection.

To overcome these limitations, the idea is to use a tool able to capture the semantic of the image and that doesn't impose model restrictions: *deep learning*. Deep learning is a field of *machine learning* that received a steadily increasing attention due to the long story of successes in almost every branch of

Figure 2.5: Sun's reflection can wipe the marks from the image.

(a) Original image. (b) Wrong detection on the right mark.

Figure 2.6: The pattern on top of the right curb is clearly stronger than the actual lane marking.

computer science. Image classification, image captioning, image segmentation, machine translation, speech recognition, speech synthesis and game playing are

just some examples of applications where deep learning established the state-of-the-art performance. There exist an exhaustive literature on deep learning applications and for a complete reference see [49].

The following section describes how I employed this flexible tool to achieve reliable lane detection.

2.3 CenterNET

The main limitations of the classic approaches to lane detection, are related to the use of lane markings as indicators. They represent the most obvious hand-designed feature for such problem, and one that should be easily identified in the image. They're not, however, the only indicator of the road lanes and there are other visual features that could help in the process. For this reason, the ability of CNN[2] to automatically learn the relevant features present in the input is very valuable. Convolutions are the standard way to process images in neural networks, and they laid the foundations of modern image processing [50].

Leveraging a CNN architecture I designed the CenterNET, which stands for Centerline NETwork, to predict the position of the ego-lane center in the image. The following presents the detail of such application.

2.3.1 Dataset

Deep learning application are known to be data hungry: to properly exploit the high capacity of the model, a big set of labeled examples is needed. Manual labeling is a very expensive and time consuming process. For this reason I designed a convenient semi-automatic annotation process. The choice of predicting the *center* of the ego lane, comes with the important advantage that its position can be recorded just driving the car in the middle of the road.

Projecting the future position of the car on the current image, enables the

[2]Convolutional Neural Network

production of annotated examples in a very cheap and fast way. To compare this semi-automatic annotation with manually labeled examples, given that both solutions require "car time" to record the images, the system takes 7ms to process a sample while a fast human annotator requires 4–5s. The actual annotation time is about 650x faster and even the quantity of images annotated with the same amount of "car-time" can be 2–3x as will be detailed later. This is the key that enabled the readily deployment of accurate ego-lane detection with CenterNET.

Input and output

The input to the network is a gray-scale image of the road as seen by the vehicle. The image comes from the right camera of the FFN (Far Front Narrow) stereo couple installed on DEEVA. Figure 2.7 shows the position of the

(a) Position of the camera. (b) Closeup.

Figure 2.7: The right camera of the FFN stereo couple is used to acquire the images in the dataset.

camera, behind the windshield, facing the road ahead. The camera is an `IDS UI-5242LE-C-MB` with an `e2V EV76C560ACT 1/1.84"` CMOS optical sensor and 12Mpixel 8mm lens. An example of image produced by the camera is in figure 2.8. The lens distortion is assumed to be removed with the appropriate

Figure 2.8: Example of image coming from the FFN-Right camera.

LUT[3] [51].

To reduce the computational cost, the original 1280x880 pixels image is down-sampled to the size of 300x200 and, to focus the processing on the road, the top 100 rows are discarded, reaching the final size of 300x100. Image 2.9 show the initial image, the intermediate down-sampled image and the final cropped result. This image is the input fed to the network.

The intended output of the network is the position, in the image, of the ego-lane center. Once the image projection of the center is computed, the label contains, for each row with a projected point, the normalized horizontal coordinate of such point. The coordinate are expressed in the interval $[0,1]$ with 0 being the left and 1 the right side of the image. Image 2.10 shows a

[3]LookUp Table

final training sample with the label overlaid.

(a) Original image.

(b) Downsampled version.

(c) Final cropped image.

Figure 2.9: Transformation process of the original image to obtain the final input for CenterNET.

Figure 2.10: Final CenterNET sample with overlaid annotation.

Semi-automatic annotation

To develop this semi-automatic annotation I leveraged the availability of the car position ground truth through the high precision INS installed on DEEVA. The described operations are performed, in post-processing, on the data recorded with VisLab's integration framework GOLD [52].

The process requires what I call a "annotation lap" in which the car is driven along the intended road, staying in the approximate middle of the lane. Note here that, since this is made by a human driver, the recorded track will not be at the exact center of the lane. To account for this imprecision, multiple annotation laps are taken for every recording session in such way that the multiple small variations from the true center will hopefully compensate each other. One thing has to be noted here, since multiple annotation laps can be taken for each recording session, and they express the centerline position in a global fixed reference system, every run can be processed with all the annotation laps. This doesn't just compensate for annotation inaccuracies, but it also means that N annotation laps will provide Nx valid number of training samples for the same amount of car time.

The annotation lap provides the GPS coordinate of the center of the car's ego-lane. The GPS is also used to know the exact position at which every image is taken, figure 2.11. To proceed with the point projection, another important ingredient is needed: an accurate camera calibration. The FFN-Right camera is supplied with intrinsic and extrinsic calibration parameters [53]. The calibration of the camera provides access to the projection matrix $\mathbf{K} \in \mathbb{R}^{3\times3}$ and the coordinate conversion $\mathbf{T} \in \mathbb{R}^{3\times4}$. Both matrices work in homogeneous coordinate [54], $\mathbf{T}$ transform the coordinates of point expressed w.r.t. the INS (which is the reference point for the extrinsic calibration) to the camera reference frame (losing the trailing '1' which is irrelevant for the following transformation), while $\mathbf{K}$ applies the projection equations for the pin-hole model obtaining the image coordinate of the given world point. [55] provides an extensive coverage on the subject of calibration and projection matrices. Image 2.12 shows the reference systems involved and provides a

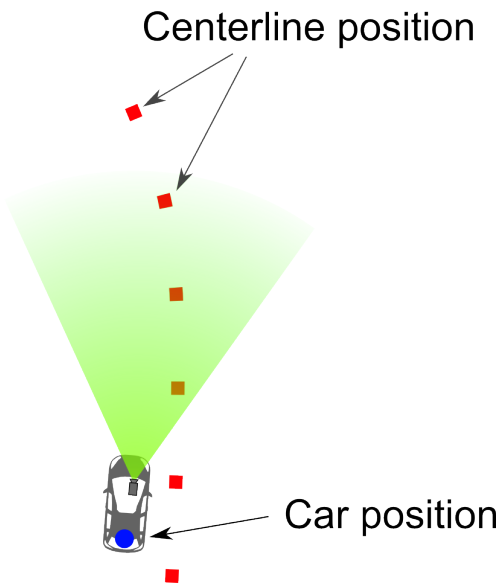


Figure 2.11: The high precision INS provides localization of the recorded images with respect to the centerline coming from the annotation lap.

schematic summary of the projection process.

For every image to be annotated, the GPS coordinate of the centerline must be converted in the current INS reference frame. Th GPS coordinate are given complying with the `WGS84` standard and the conversion is executed through the intermediate transformation to ECEF[4] coordinate, a Cartesian reference system centered in the Earth's center of mass. The conversion from GPS coordinate to ECEF involves the projection on the ellipsoid approximation of the Earth and will be indicated with:

$$p_{\mathrm{e}} = \mathrm{ecef}(p_{\mathrm{w}}) \tag{2.1}$$

where p_{w} are the `WGS84` coordinate of the centerline's point p and p_{e} are the coordinate of the same point in the ECEF reference system. The conversion equations are well know and can be found in [56]. Having the ECEF coordi-

[4]Earth Centered Earth Fixed

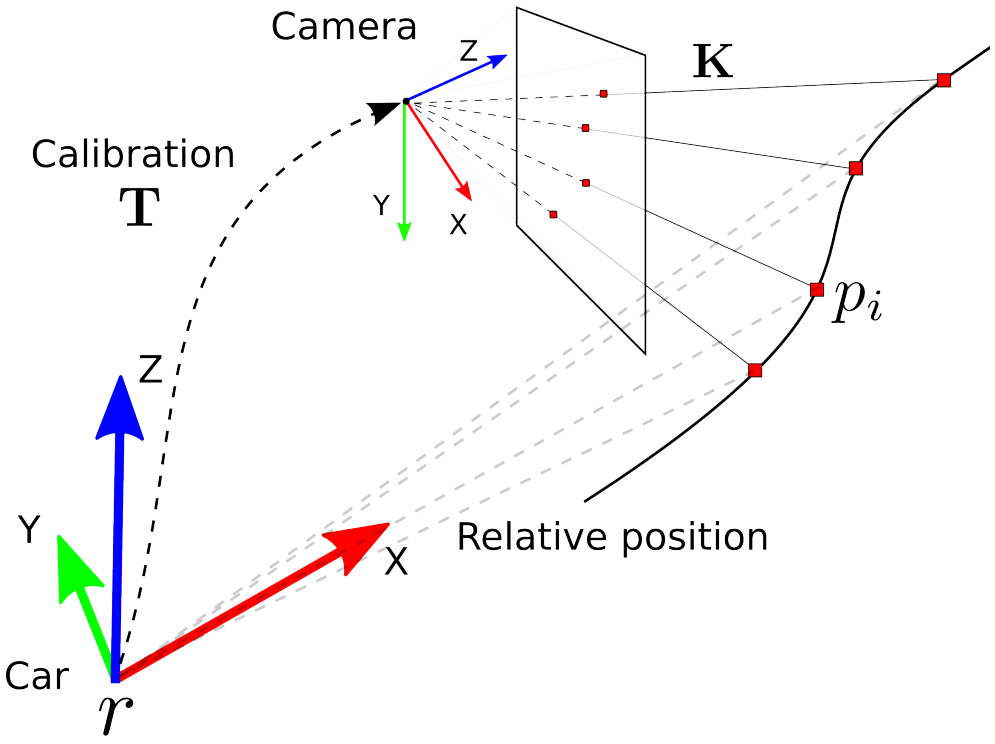


Figure 2.12: Summary of the projection process.

nate of the centerline's points $p_{e,i}$ and the current reference point (the car's position) r_e, the relative coordinates are obtained with a simple difference, obtaining ${}^{r}p_{e,i}$. To finally recover the navigation ENU[5] coordinates p_n, the ECEF points must be rotated about the Z and Y axis of the longitude and latitude angles with the matrix ${}^{n}_{e}\mathbf{R}$. See [56] for a detailed coverage on this topic. The ENU reference system has the axis aligned with the cardinal direction, so the heading of the car (the angle relative to the North) must be considered with

[5]East North Up, indicating the direction of the X, Y and Z axes respectively.

a further rotation about the new Z axis with the additional matrix $\bar{\mathbf{R}} \in \mathbb{R}^{3\times3}$.

For a precise projection, also the attitude of the car relative to the ground plane must be compensated. Bump and imperfection in the pavement cause the car to pitch and, when turning, the suspension springs allow the car to have a roll angle. The instantaneous rotation angles are also provided by the INS platform and these new rotations about X and Y are incorporated in the matrix $\bar{\mathbf{R}}$. Indicating with ρ, θ and ψ the roll, pitch and yaw angle respectively, the final form of the matrix $\bar{\mathbf{R}}$ is:

$$\bar{\mathbf{R}} = \begin{pmatrix} \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ \sin(\rho)\sin(\theta)\cos(\psi) - \cos(\rho)\sin(\psi) & \sin(\rho)\sin(\theta)\sin(\psi) + \cos(\rho)\cos(\psi) & \sin(\rho)\cos(\theta) \\ \cos(\rho)\sin(\theta)\cos(\psi) + \sin(\rho)\sin(\psi) & \cos(\rho)\sin(\theta)\sin(\psi) - \sin(\rho)\cos(\psi) & \cos(\rho)\cos(\theta) \end{pmatrix} \tag{2.2}$$

Figure 2.13 display a case where a bump cause the car to pitch sensibly, note that without proper pitch compensation the annotation is projected far above the road plane.

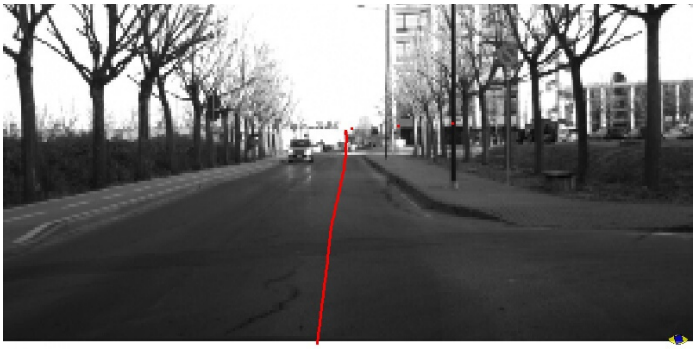
Putting everything together, the final transformation is:

$$^{\text{image}}p = \begin{pmatrix} u_p \\ v_p \end{pmatrix} = \mathbf{K}\,\mathbf{T}|\bar{\mathbf{R}}\,{}^{\text{n}}_{\text{e}}\mathbf{R}\,(\text{ecef}(p_{\text{w}}) - \text{ecef}(r_{\text{w}})) \tag{2.3}$$

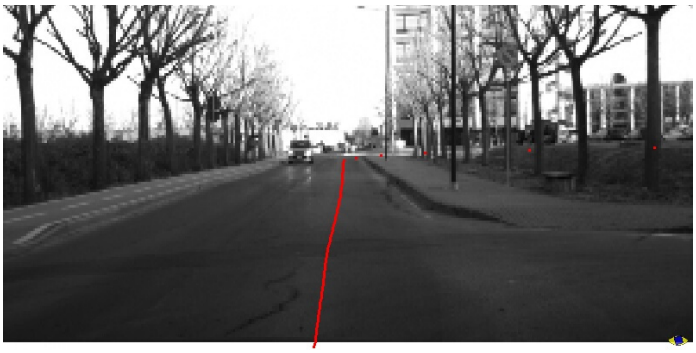
where the vertical bar | indicates the extension of the vector to homogeneous coordinate.

With the detailed projection process, all the centerline points in front of the current position are converted in image coordinates until the projection ends outside of the image size. Since the projection uses the original camera parameters, it is equivalent to projecting the points in the non-scaled version of the image. The horizontal and vertical scale factors η_W and η_H, are applied to the u-v coordinate of the projected points:

$$\begin{pmatrix} \widetilde{u}_p \\ \widetilde{v}_p \end{pmatrix} = \begin{pmatrix} u_p/\eta_W \\ v_p/\eta_H \end{pmatrix} \tag{2.4}$$



(a) Centerline projection without pitch compensation.



(b) Centerline projection with pitch compensation.

Figure 2.13: Example of pitch compensation.

The final u-coordinate (corresponding to the image column) of the projected points are obtained averaging the values falling in the same row:

$$\bar{u} = \frac{1}{n} \sum_{p} \widetilde{u}_p \Big| \lfloor \widetilde{v}_p \rfloor = \bar{v}, \quad \bar{v} = 0, 1, \dots \tag{2.5}$$

Input distribution

One thing that must be taken with extreme care is the distribution over the input images. Since the output of this component will influence somehow the driving behavior, there exist the possibility of a catastrophic feedback loop. If

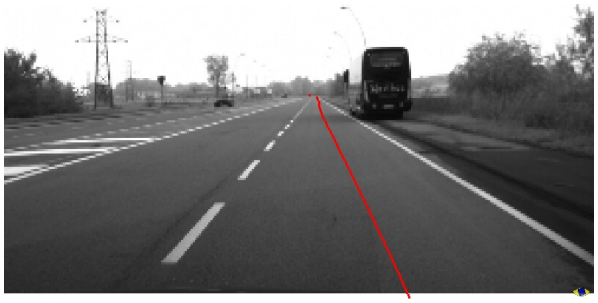
the CenterNET detects a centerline which is not accurate, the car might react steering toward the incorrect estimated position. This can take the camera to see images different than the one shown by the human driver, which is likely to drive close to the center of the lane. In turns, the CenterNET is unlikely to properly react to inputs that lay outside the training distribution, so the error will grow resulting in a completely faulty driving behavior. This is an example of *distribution mismatch*, a well known problem in machine learning applied to agent that can act and modify the environment.

This problem must be taken into consideration and the key to the solution is a smart use of the auto-annotation process. Since the same annotation can be used across different recordings, the only run constrained to be driven in the middle is the annotation lap itself. The other runs are allowed, encouraged in fact, to explore other point of view within the same lane.

Following this precaution, every recording session captured the same course in multiple ways. Other than the annotation lap, the runs are taken driving in the left/right side of the lane or oscillating in the middle in order to provide even extreme angles on the road. Figures 2.14 shows the same road from different perspective, as present in the training data.

Driving the car with an offset from the center also prevents the CenterNET to overfit the most likely centerline position, which for the low rows would always be at the center regardless of the shape of the road.

This smart choice for the input distribution stabilize the network and helps the generalization process, as the CenterNET is actually forced to find explanatory cause for the different centerline position. The next section describe another technique to further help with stabilizing the network.



(a) Car on the left, aligned with the road.

(b) Car steering left.

(c) Car on the right, aligned with the road.

(d) Car steering right.

Figure 2.14: Examples of different input distributions.

Dataset augmentation

Dataset augmentation is a popular technique for increasing the training set size and improve the generalization of the network. It consist in modifying the training samples in a way that doesn't affect their label (or the effect is know and can be compensated). The benefits are twofold. First, the overall size of the training set is increased as it contains both the original and the modified samples. Secondly, if the modifications don't affect the label, the network should learn to be invariant to the injected perturbations.

I used aggressive dataset augmentation to further stabilize the CenterNET.

The first augmentation technique I used is noise injection. Since the noise in the image should not modify the predicted position of the centerline, this simple technique can effectively decrease the network sensibility to small variations in the input. Qualitatively, this technique alone was able to appreciably reduce the output oscillations due to the sensor's noise during the actual deployment on the car.

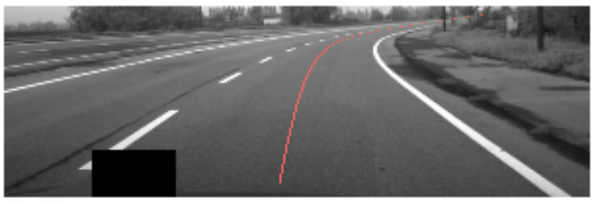
Another, more aggressive, augmentation technique I explored is the simulation of "missing" image region, overwritten with black squares. This perturbation loosely simulate the effect of image occlusion and encourage the CenterNET to exploit the remaining features.

Finally, to further encourage the use of context beside the lane markings' edges also a Gaussian blur is used as an augmentation technique.

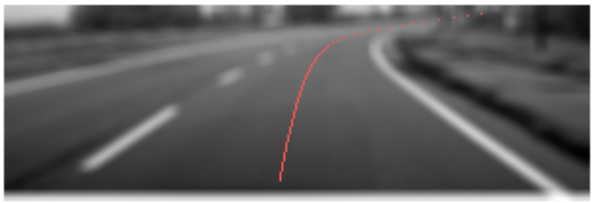
Figure 2.15 shows the effect of the employed augmentation techniques on a training example.

(a) Original example.

(b) Noise injection.



(c) Black square painting.



(d) Gaussian blur.

Figure 2.15: Examples of the applied augmentation techniques.

Statistics

The generated dataset contains a total of 184059 labeled examples. The scenario distribution in the dataset is the following:

Scenario type	**# of examples**	**relative %**
Rural	61175	33.2%
Highway	51737	28.1%
Urban	49204	26.7%
Ring road	21466	11.7%

Of this examples, 1929 constitute the test set (about 1%) with the same scenario distribution. Other notable statistics are:

Feature	**relative %**
Left lane	3.9%
Nighttime	1.0%

Figure 2.16 provides examples for each scenario.

(a) Rural.

(b) Highway.

(c) Urban.

(d) Ring road.

Figure 2.16: Examples of the different scenarios present in the dataset.

2.3.2 Architecture

The network needs to process an input image to predict the centerline position of the ego-lane. The standard way to process images in neural networks is through a series of convolutions, so the CenterNET follows the typical structure of CNN.

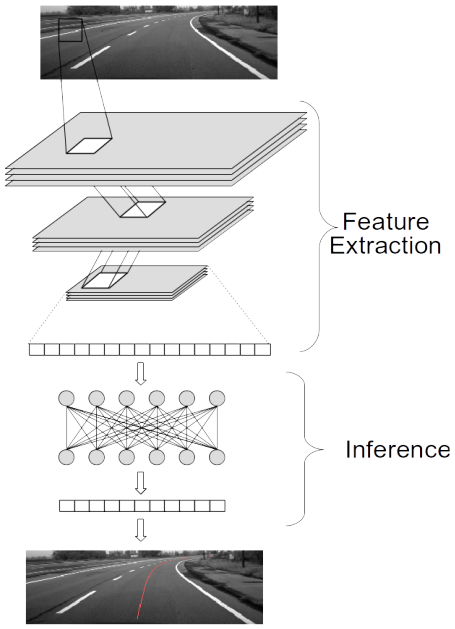


Figure 2.17: Schematic CenterNET architecture.

The architecture can be roughly divided in two sections: *feature extraction* and *inference* (figure 2.17). The feature extraction part is composed by a series of convolutions. The image is transformed in several features maps with smaller dimension and, at the final stage, a linear feature vector is obtained. The inference part takes this feature vector and, using fully connected layers,

produces the final output.

The next sections provide a detailed description of each part.

Feature extraction

The feature extraction part is responsible, as the name suggests, for extracting a vector of relevant features from the image. This is performed with a series of convolutions that shrink the image, but increase the number of channels. This intermediate images are called feature maps and should contain relevant information, preserving the spatial relations.

The usual convolutional layer (used for examples in classification tasks) is composed by the convolution, followed by a transfer function and a pooling layer [57]. The pooling layer is used to shrink the image size and, in the case of image classification, provide some translation invariance with respect to the pattern of interest. Since CenterNET is not performing classification, but actual detection I decided to exclude pooling layers from the architecture and shrink the image using convolution with strides greater than 1. The removal of pooling layers has another practical advantage: it reduces the size of intermediate data the network needs to store. Making the network smaller in memory allows bigger batches to be processed at the same time, effectively speeding up the training process.

The results are provided for 2 different variants of feature extraction: *Type-I* and *Type-II*, detailed in table 2.1 and 2.2 respectively. Both Type-I and Type-II include a dropout layer [58] to regularize the network.

#	**Layer type**	**Parameters**
1	2D convolution	kernel=5×5, channels=$1 \rightarrow 12$, stride=$1, 1$
2	leaky ReLU	leak=0.1
3	2D convolution	kernel=5×5, channels=$12 \rightarrow 12$, stride=$1, 1$
4	leaky ReLU	leak=0.1
5	2D convolution	kernel=5×5, channels=$12 \rightarrow 12$, stride=$1, 1$
6	leaky ReLU	leak=0.1
7	dropout	p=0.3
8	2D convolution	kernel=5×5, channels=$12 \rightarrow 12$, stride=$2, 2$
9	leaky ReLU	leak=0.1
10	2D convolution	kernel=5×5, channels=$12 \rightarrow 12$, stride=$2, 2$
11	leaky ReLU	leak=0.1
12	2D convolution	kernel=5×5, channels=$12 \rightarrow 4$, stride=$2, 2$
feature vector of 1056 elements		

Table 2.1: Network architecture for type-I feature extraction.

#	**Layer type**	**Parameters**
1	2D convolution	kernel=5×5, channels=$1 \rightarrow 64$, stride=$2, 2$
2	leaky ReLU	leak=0.1
3	2D convolution	kernel=5×5, channels=$64 \rightarrow 64$, stride=$2, 2$
4	leaky ReLU	leak=0.1
5	dropout	p=0.3
6	2D convolution	kernel=5×5, channels=$64 \rightarrow 64$, stride=$2, 2$
7	leaky ReLU	leak=0.1
8	2D convolution	kernel=5×5, channels=$64 \rightarrow 64$, stride=$1, 1$
feature vector of 1050 elements		

Table 2.2: Network architecture for type-II feature extraction.

Inference

The inference part makes the actual prediction of the centerline position using the extracted feature vector.

The inference block produces a fixed-size centerline prediction, meaning that the final layer outputs a vector of N values representing the position of the centerline on N rows of the image selected in advance. To train such a network, only the training samples containing a centerline point for *all* the rows are used. Note that the number of rows occupied by the centerline can vary considerably with the scenario. The figure 2.18 show two examples of different centerline length found in urban and highway scenarios.

(a) Urban environment. (b) Highway environment.

Figure 2.18: Examples of different centerline length due to different scenarios.

Finally, batch normalization [59] is applied to the initial feature vector to speedup the learning process. Table 2.3 details the structure of the inference block, where N is the size of the feature vector and M is the size of the output which is the number of rows for which the network provides a prediction. Using fully connected layers, this section contains the vast majority of the network's parameters. Assuming an output dimension of 66 rows, the inference block contains 99.95% of the 1608576 parameters for Type-I and 99.29% of 1804399 for Type-II.

#	Layer type	Parameters
1	leaky ReLU	leak=0.1
2	batch normalization	
3	linear layer	size=$N \rightarrow 1000$
4	leaky ReLU	leak=0.1
5	linear layer	size=$1000 \rightarrow 500$
6	leaky ReLU	leak=0.1
7	linear layer	size=$500 \rightarrow M$

Table 2.3: Inference architecture.

2.3.3 Training

Loss function

The loss function used to train the CenterNET is a weighted variant of the standard MSE[6] loss. Since the same detection error at different rows corresponds to different errors in the world space, I decided to assign a different weight to every row. In particular, the top rows of the image correspond to road points that are further away, so they're assigned a bigger weight. To obtain a detection error that relate directly to the error projected in the world and measured in meters, the road is assumed to be a flat plane, corresponding to $Z = -h$, where h is the height of the INS reference system from the ground. With this simplifying assumption, knowing the camera calibration, it is possible to compute the IPM [46] transformation and know the world coordinate of the image points projected on the road. Leveraging the IPM, every row is assigned a weight equal to the Y displacement of the projected point, relative to a 1 pixel displacement in the image, as shown in figure 2.19.

So the final form of the loss function is:

$$E(\hat{y}, y) = \sum_{i=1}^{N} \left[w_i \left(\hat{y}_i - y_i \right) \right]^2 \tag{2.6}$$

[6] Mean Squared Error

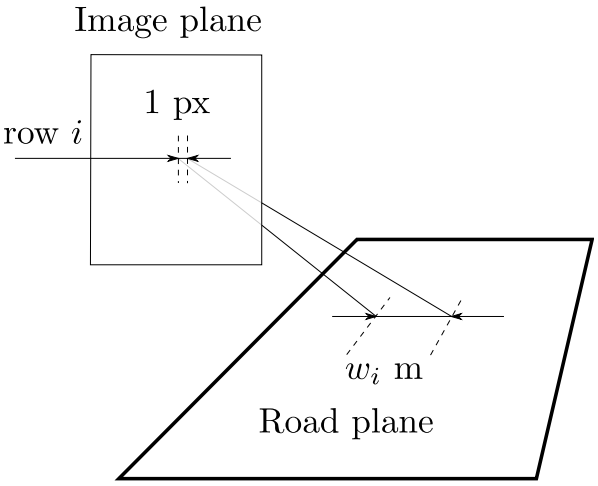


Figure 2.19: The weights are the Cartesian displacement corresponding to a 1pixel error in the image.

where $\hat{y} \in \mathbb{R}^N$ is the output produced by the network and $y \in \mathbb{R}^N$ is the true label.

Optimization

The followed training procedure is the same for both Type-I and Type-II.

The network's parameters are optimized with the ADAM [60], a first-order gradient based optimization algorithm for stochastic function. ADAM uses adaptive estimation of first and second order moments with bias correction, which are used to adjust a per-parameter learning rate vector. The value of the algorithm's hyper-parameters are:

Hyper-parameter	**Value**
initial learning rate	10^{-4}
β_1	0.9
β_2	0.999

The gradient is stochastically computed with mini-batches of 512 samples and the norm is clipped to 10^6 to prevent "gradient-explosion" and instability.

The network is trained for a maximum of 500 epochs, using early stopping on a validation set composed of the 5% of the training examples. The validation error is computed every 20 iterations and the training is halted when no improvement is observed for 50 consecutive checks.

2.3.4 Results

The table 2.4 report the error statistics on the test set. The "global" error is obtained averaging over all the rows, while the various distance ranges consider only the rows that, on the road plane, correspond to that longitudinal distance. The error is expressed in meters and is obtained with the weights detailed in 2.3.3. This measure reflects directly the reliability of the detection in terms of autonomous navigation, which is the most relevant in the context of this research. From the results emerges that the two architectures perform similarly, but Type-II is consistently able to achieve lower average error and variance, while Type-I yields the minimum max error.

Note that the average error for Type-II is about 10cm, which is in the precision threshold that can be expected from the human-driven annotation lap.

Due to the novelty details of this application, there are no state of the art results to compare with. The presented state of the art in lane detection provides only qualitative results and they solely concern lane markings detection. Moreover the detection error in the image is loosely related to the quality of the detected path, which is also not directly available, while the reported results for CenterNET are immediately relevant for the autonomous navigation task.

The images 2.20 show some good results taken from the test set, while in 2.21 there are examples of prediction failures. In some cases, while being different from the ground-truth label, the network's result is acceptable, like in 2.22.

A notable result is the ability of the network to produce reasonable predictions in adverse lighting conditions. Images 2.23 shows example with hard

		Type-I	Type-II
<10m	avg	0.080	**0.071**
	max	**0.72**	0.91
	std	0.085	**0.079**
10-20m	avg	0.11	**0.095**
	max	**1.2**	1.3
	std	0.12	**0.10**
20-30m	avg	0.21	**0.17**
	max	**2.9**	3.0
	std	0.26	**0.22**
Global	avg	0.11	**0.097**
	max	**1.1**	**1.1**
	std	0.11	**0.097**

Table 2.4: Training results.

shadows, sun reflection and partial blinding. In the same way, also in the nighttime environment (about 1% of training examples) the network shows consistent performance, figure 2.24.

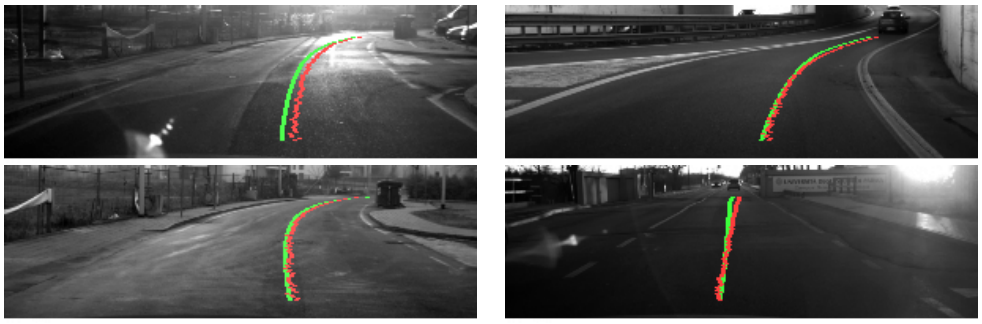


Figure 2.20: CenterNET good predictions (red) with annotation (green).

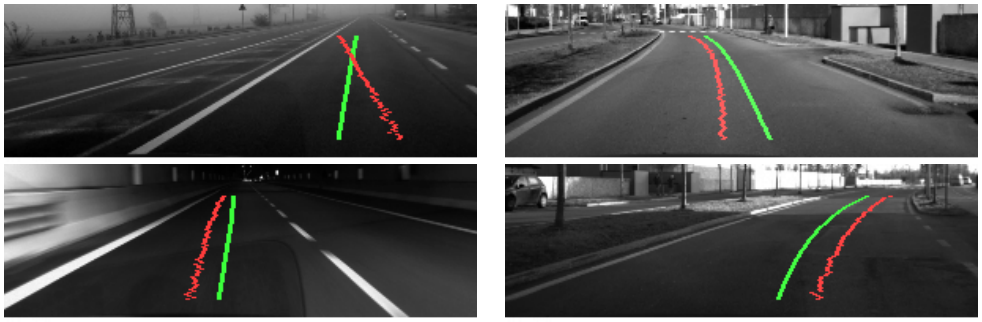


Figure 2.21: CenterNET prediction failures (red) with ground truth (green).

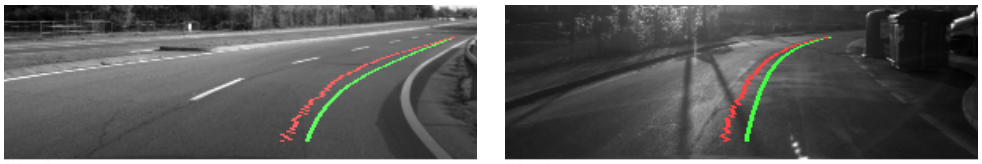


Figure 2.22: CenterNET acceptable predictions (red) with ground truth (green).

Figure 2.23: CenterNET predictions in adverse lighting (red) with ground truth (green).

Figure 2.24: CenterNET in nighttime environment (red) with ground truth (green).

2.3.5 Conclusions and future development

CenterNET is a novelty application of CNN in the context of autonomous driving and a new approach to ego-lane detection. The network is able to perform detection within a satisfying accuracy in challenging environments, without being constrained to the presence of visible lane markings. It also handles arbitrary road shape, without limitations to predefined models. Moreover, the newly introduced semi-automatic annotation procedure enables fast and efficient production of training examples, making possible to handle arbitrary scenarios within a short development time.

CenterNET has been successfully deployed on DEEVA, helping with the task of autonomous navigation in difficult environment.

There are however some limitations that must be highlighted.

Figure 2.25: Example of driving case where the image contains the road only in the very bottom rows.

Predicting a fixed-size ego-centerline has the advantage of making the architecture simple and the problem relatively easy to learn. A major drawback is that, at test time, there is no mean to establish if the centerline is actually present in the image rows. While normal driving condition will, most likely, place the centerline in the image, if the car faces something that's not the

road (for example during a tight turn), the network will still make a prediction for the rows selected at training time. Image 2.25 shows an example of such condition, occasionally occurring in some driving scenarios. To overcome this limitation, the network should be able to dynamically adjust the prediction range based on the image content. The typical deep learning networks have a static architecture, meaning that the "shape" of the output is fixed when defining the network. To allow a dynamic prediction range, there are two main possible approaches.

One solution is to move from a pure feed forward architecture to a mixed feed forward and recurrent architecture. In particular, the fully connected layer of the inference block should be replaced with a one-to-many RNN[7][61]. This RNN would be trained to output a sequence of prediction, one for every valid row of the image, conditioned on the extracted feature vector. These kind of RNN are referred to as one-to-many because they output an arbitrary sequence of values for a single input [62]. The assumption here is that the valid rows in the image are always a contiguous interval starting from the bottom of the image; in other words when the image contains an invalid row, moving toward the top, also the remaining ones are considered to be invalid.

An alternative approach is to keep a pure feed forward architecture, but augment the network with an additional output vector that express the probability of the rows to be valid. This additional vector would be used at test time to assess for which rows the network prediction is valid. The potential advantage of this solution is that it avoids a recurrent architecture that is known to be more difficult and less efficient to train.

Another limitation of the proposed CenterNET architecture is that it is tied to the concept of ego-lane. While being relevant in the context of autonomous navigation, the concept can be blurry, for example when changing lane, and in the general case it would be helpful to have information about all the available lanes. To overcome this limitation it could be possible to change the inference block with a series of upsampling convolutions [63] (also known as full

[7]Recurrent Neural Network

convolutions or fractionally strided convolutions) that produce a 2 dimensional output. This output image could be interpreted as the probability of a given pixel to belong to a centerline of some lane. Using the same semi-automatic annotation, with multiple annotation laps, it could be possible to train such a network to predict the center of all the lanes present in the image.

The proposed solutions would allow to overcome the highlighted limitations, delivering a more flexible and powerful CNN based lane detection.

Chapter 3

Data Fusion: Road Path Estimation

3.1 State of the art

The previous chapter introduced a novelty approach to lane detection that tries to overcome some limitations of classic methods. While the system shows good accuracy over a variety of scenarios, the output is still affected by noise and it's not suitable for a direct use as a navigation reference. The presence of noise is a common problem to every detection algorithm and the solution commonly adopted is to add a tracking/filtering phase of the obtained results.

The tracking phase can be performed at different levels. For example [64] tracks the final lane model's parameters, while [45] performs tracking of the single lane marking proposals. The tracking technique is selected according to the specific needs. If the tracking is performed separately for each instance, or if the problem has a unimodal distribution of parameters, some flavor of Kalman filter is used [65, 66, 67]. When multiple hypothesis need to be tracked at the same time, or the distribution is multinomial, the particle or PHD filter can be used [68, 69].

A careful tuning of tracking parameters can greatly improve the stability

and smoothness of the output, however relying on a single source of information poses serious limitations on the overall reliability. As denoted in section 2.1, the vast majority of visual lane detection algorithms are based on lane markings, and section 2.2 already highlighted some situations in which these features are insufficient to identify the ego-lane. Moreover, all the vision-based algorithms are susceptible to difficult lightning conditions and share the same failure modes. To improve the robustness of the ego-lane estimation, the system should integrate information coming from various visual cues and even from sensors with different failure modes. A popular choice in the literature is to use the LIDAR to identify occlusion due to obstacles [70], estimate the ground roughness to help the segmentation task [71, 72] and detect curbs and berms [73]. RADAR is another sensor that can be used to detect road boundaries [74] or the road plane itself [75].

3.2 Proposed approach

Without a centimeter level absolute localization, recorded lane tracks are not a viable solution, so the only navigation reference the system has is the one that can be perceived with the on-board sensors. For this reason, in our context, a stable and reliable road path estimation is of paramount importance for autonomous navigation. Unlike most of the literature work that focus on understanding the topology of the road, counting the lanes and estimate which one is occupied by the ego-vehicle, I focus on getting a quantitative measure of the ego-lane position that can act as a navigation reference for the path planner, like in [76]. This application imposes rigid requirements on the reliability and stability of the estimation to achieve a safe and smooth navigation.

To achieve the required solution quality I propose a multi-sensors fusion approach framed in the context of moving horizon estimation [77]. The fusion algorithm leverage all the pertinent features available on the DEEVA's perception system: lane markings, CenterNET, radar obstacles, barriers and curbs.

Lane markings : the lane boundaries are identified with a lane marking detector based on [45]. Using a IPM transform, the markings detected in the image are projected on the road plane and their Cartesian coordinates, with respect to the car body, are computed. As already discussed, various condition can hinder the marking detection and this feature is not available in all the roads. Figure 3.1 provides some examples of output from the lane marking detector.

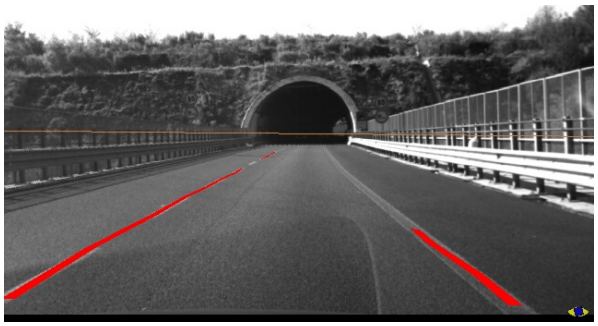


Figure 3.1: Lane markings detection output.

CenterNET : the center of the ego-lane is detected with the newly introduced CenterNET. Similarly to lane markings, the Cartesian coordinates of the centerline are recovered with the IPM transform. CenterNET exploits many different visual features, but it's subject to the camera's failure modes. Moreover, being a machine learning application, its stability can only be statistically assessed and some unexpected input can induce detection errors. Some output examples are shown in section 2.3.4.

Barriers, curbs : the disparity image from the FFN couple is used to gain further information about the road border. Vertical contiguous barriers and curbs are detected and used as indicators of the road boundaries. Using the disparity information, these features natively provide the Cartesian coordinates. The performance of these applications is directly related to the quality of the disparity information. Adversarial

lightning conditions and lack of textures can prove a serious challenge for stereo matching algorithms with consequent degrade of the detection accuracy. Examples of barrier and curb detections are depicted in figure 3.2.

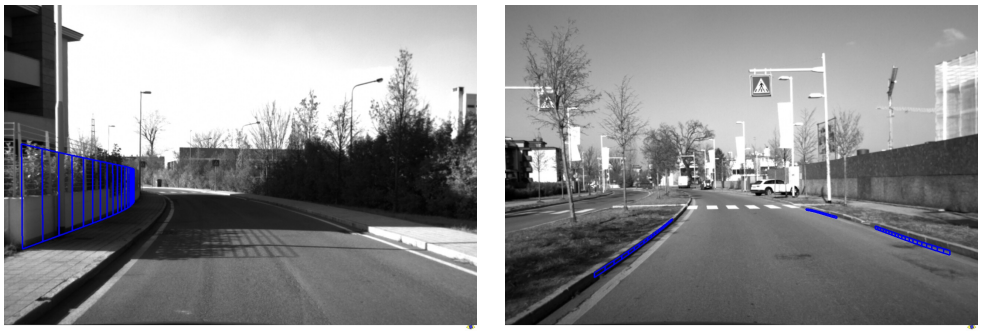


Figure 3.2: Barriers and curbs.

RADAR obstacles : to provide robustness to the camera's failure mode, the RADAR information is integrated. RADAR provides a very reliable detection for some kind of road boundaries. Light poles, street signs and, occasionally, trees are good indicators of the road border in urban environment. In the highway scenario the guardrail marks a neat boundary and when driving tunnels their walls play the same role. While being reliable, the RADAR information is also very poor in semantic and should be used with care especially at long detection range. Figure 3.3 shows some examples of road boundaries properly detected from the RADAR.

All the mentioned measures are, somehow, affected by noise, so the fusion algorithm must properly deal with detections' imprecisions. Assuming a zero mean noise in the data, the proposed approach utilizes a sliding window buffer of measures to perform refined estimations. The idea is that, averaging multiple measures, the small variations from the true value will compensate each other.

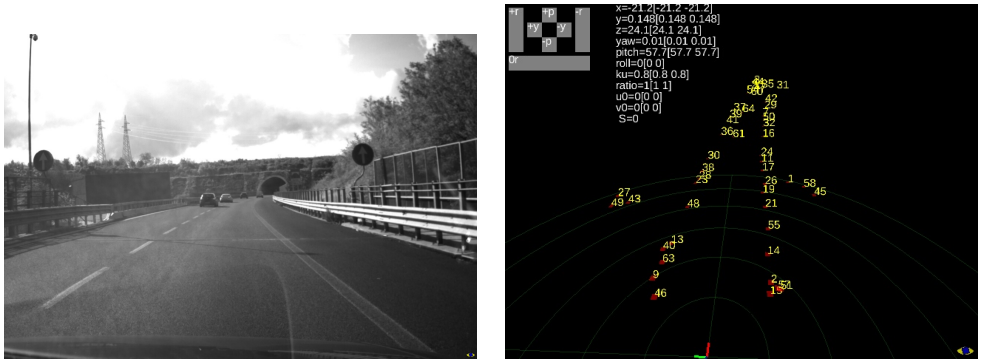


Figure 3.3: Image and corresponding radar echoes.

Using this window of buffered measures, the fusion algorithm estimates any number of valid lane models using little prior knowledge about the road. The optimal estimation is provided in closed form and the procedure is independent form the actual model used to represent the lane, that can be chosen in the broad family of linear basis models.

The following sections detail the developed fusion algorithm, providing also results for some actual use cases.

3.3 Algorithm

This section introduces the fusion algorithm, starting with the general operation flow and then detailing the relevant sections.

Before starting the description, it's important to introduce some basic concept used in the algorithm: the involved reference systems, the lane models and the road reference.

The algorithm works in 2D, constraining the road to lay on a plane, and uses two different reference frames: *body* and *navigation*, shown in figure 3.4. The body frame is attached to the vehicle and it's the one used to express the sensors' calibration. This means that the Cartesian measures coming from the detection algorithms are natively expressed in this reference system. The

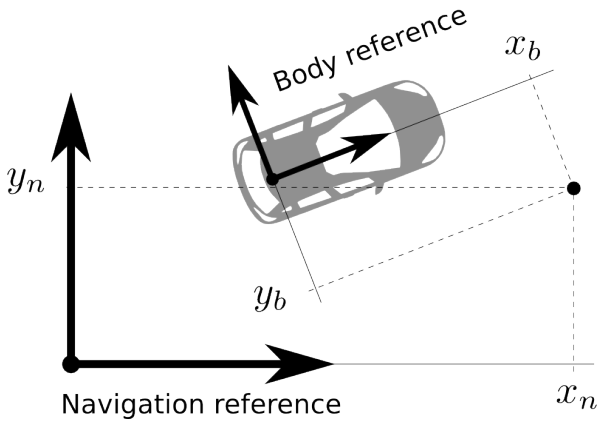


Figure 3.4: Navigation and body reference systems.

other is the navigation frame, a global reference system the position of which is fixed with respect to the world. Note that having a global frame doesn't mean that the system needs a precise global localization. The only requirement for the navigation frame is to be locally consistent: in a short period of time, the estimated movement should be as close as possible to the real one. The estimation of the navigation pose can come from the integration of odometry and IMU signals; long-term divergence is not a concern. The meaning of "locally" and "long-term" depends on the temporal window used for the fusion process: it will be assumed that the movement estimation is consistent within the temporal span of the buffered measures.

The road is considered to be a collection of lanes, so the estimation is focused on the individual lane models. A lane model is assumed to be a function of the body coordinates. This choice reflects and exploits the regularities of the road shapes as perceived by the vehicle, allowing fairly simple models to represent a great variety of situations. Moreover, the lane model is constrained to belong to the family of linear basis functions:

$$y = f(x;\sigma) = \sum_{i=1}^{n} \sigma_i \phi_i(x) \tag{3.1}$$

where ϕ_i are the basis function, assumed to be twice differentiable. With this

model family, computing the optimal model reduces to estimating the optimal coefficient vector $\boldsymbol{\sigma}^* = (\sigma_1^*, \ldots, \sigma_n^*)^\top$.

The basic input of the algorithm is referred to as *road reference*: a collection of Cartesian 2D points that can be expressed with respect to either body or navigation frames. As an example, the output of the lane detector is a series of road references, one for each marking, every of which contains multiple 2D points that represent the body coordinates associated with that marking. During the processing, additional informations are attached to the road reference, like the source algorithm (whether it is lane marking detection, CenterNET, etc. . .), the buffering time and possible associations with the estimated models. For clarity of notation, a short label is assigned to every detection algorithm:

lm : lane markings detection.

cn : CenterNET.

rd : RADAR.

br : barriers detection.

cr : curbs detection.

and $T = \{\text{lm, cn, rd, br, cr}\}$ is used to refer to the collection of all the sources.

When fed as input, the road references are assumed to be expressed in body reference frame relative to the current car position, also provided as input to the algorithm. Note that if the detections comes from different time-points, the coordinate of the relative road references must be transformed to account for the car movement as estimated from the change of navigation coordinates.

The first operation on the input data, called preprocessing, is aimed at reducing the number of false measures being later considered. Once cleaned, the new road references are transformed into the navigation frame (using the provided car position) and inserted into the road reference buffer, saving the

original source and the time of insertion. Road references older than a configurable time threshold are removed from the buffer.

At this point, association between the estimated lanes and the buffered road references are searched for. Each model is refined with the associated measures or gets discarded if no valid associations are found. Road reference with compatible associations to the same lane model are fused together to improve stability.

Once the refinement is completed, or when there are no estimated lanes, the road references from lane markings are processed searching for couples of markings that can generate new valid lane models.

The final step consists in determining if there's a valid ego-lane and computing the output for the planning system.

The procedure is detailed by algorithm 3.1 and the following sections explain in details the relevant steps.

Algorithm 3.1 Road path estimation

Data:

$L = \{l_1, \ldots, l_n\}$ ▷ estimated lane models

$B_t = \{rr_{t,1}, \ldots, rr_{t,m_t}\}$, $\forall t \in T$ ▷ buffer of road references for every type of road indicator

Input:

$RR_t = \{r_{t,1}, \ldots, r_{t,q_t}\}$, $\forall t \in T$ ▷ detections for every type of road reference

p_{car} ▷ car position in global reference

Output: the refined road path estimation

1: **for all** $t \in T$, and $r \in RR_t$ **do**
2: PREPROCESS(r)
3: convert r in global frame
4: add r to buffer B_t ▷ measures buffering
5: **end for**
6: convert $B_t \ldots$ in body reference ▷ the model is expressed in body frame
7: **for all** $l \in L$ **do**
8: UPDATEMODEL(l, p_{car})
9: REFINEMODEL($l, B_t \ldots$)
10: **end for**
11: SPAWNNEWMODELS($B_t \ldots$)
12: convert $B_t \ldots$ in global reference ▷ buffering in global frame
13: **return** COMPUTEOUTPUT(L)

3.3.1 Preprocessing

During the preprocessing, some heuristics are used to discard measures that are obviously not relevant or correspond to false detection. The filtering is based on the orientation and lateral displacement of the road reference. The lateral displacement is considered to be the y coordinate of the geometric center of road reference, while the orientation, when applicable, is the average point-to-point orientation. To properly compute the orientation, the points are sorted in ascending value of x. These values are confronted with loose thresholds that eliminate the measures only when they're very far from the car or are oriented with extreme angles. The underlying assumption is that the car is driving in the road with a coherent orientation. The actual threshold value is specific to every kind of road reference and it's a configuration value provided to the algorithm.

Remaining measures are trimmed up to a confidence range, removing point beyond this distance. The thresholds and confidence range are separate configurations for every input source in T.

Once cleaned, the new road references are transformed into the navigation frame (using the provided car position) and inserted into the road reference buffer, saving the original source and the time of insertion. Road references older than a configurable time threshold are removed from the buffer.

Algorithm 3.2 Detection preprocessing

Input:

$r = \{p_1, \ldots, p_n | p_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} \in \mathbb{R}^2\}$ ▷ road reference

$t \in T$ ▷ source type

1: **function** DISPLACEMENT(r)
2: **return** $\frac{1}{n} \sum_{i=1}^{n} y_i$
3: **end function**
4: **function** ORIENTATION(r)
5: $\begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} \leftarrow \frac{1}{n-1} \sum_{i=1}^{n-1} \begin{pmatrix} y_{i+1} - y_i \\ x_{i+1} - x_i \end{pmatrix}$
6: **return** atan2(δ_y, δ_x)
7: **end function**
8: **function** PREPROCESS(r, t)
9: $disp \leftarrow$ DISPLACEMENT(r)
10: $ori \leftarrow$ ORIENTATION(r)
11: **if** $disp > \mathrm{d}_t$ or $ori > \mathrm{o}_t$ **then**
12: discard r ▷ eliminate di road reference
13: **return**
14: **end if**
15: **for all** $p_i \in r$ **do**
16: **if** $x_i > D_t$ **then**
17: discard p_i ▷ remove measures beyond confidence range
18: **end if**
19: **end for**
20: **end function**

3.3.2 Update model

Algorithm 3.3 Model updates

Input:

l ▷ lane model to be update

p_{car} ▷ current car position in navigation frame

1: **function** UPDATEMODEL(l, p_{car})
2: $P = \{X, Y\} \leftarrow$ navigation points of l ▷ saved centerline points for l
3: convert P in body frame with p_{car}
4: $l \leftarrow$ FITMODEL(X, Y)
5: **end function**

The lane models express the y-coordinates of the centerline given the x-coordinates in the body reference system. Between consecutive execution of the algorithm the car typically moves, thus the same lane centerline (that is assumed to remain fixed with respect to the world) will be represented as a function of the body coordinates with different $\boldsymbol{\sigma}$ parameters.

For each estimated model, the navigation coordinate of the centerline are computed at the end of every execution, so the updated coefficient $\boldsymbol{\sigma}$ can be computed performing a model fitting on those point converted in the new body frame. Section 3.3.4 will detail the model fitting procedure.

3.3.3 Association and refinement

To produce a refined estimation of the road path, the existing lane models have to be associated with the buffered road references. This association process exploits the regularities exhibited by the environment. From the analysis of different scenarios, it was evident that, for a given scenario, every road indicator category is in a consistent relationship with the lanes. For example, in the urban environment the distance from the curb to the lane marking is somehow constant, and in the highway scenario the distance of the guardrail from the lanes is fixed. These regularities are important to properly exploit

Algorithm 3.4 Association and refinement

Input:

l ▷ lane model to be update

p_{car} ▷ current car position in navigation frame

1: **function** CHECKASSOCIATION(l, X, Y, t, α, ρ)
2: $\sigma \leftarrow$ model parameters of l
3: $\Delta \leftarrow f(X;\sigma) - Y - \mathbb{1}\alpha(\frac{w}{2} + \rho)$
4: **if** $\frac{1}{n}\|\Delta\|^2 \leq \mathrm{Th}_t$ **then**
5: **return true**
6: **else**
7: **return false**
8: **end if**
9: **end function**

indicators such as curbs, barriers and radar obstacles that need to be configured with a set of possible offsets from the lane border. The problem doesn't arise for lane markings and CenterNET that are, respectively, at the border and center of the lane.

The association procedure checks if the considered model is compatible with the road reference points translated with a valid offset. The translation if defined by two parameters: $\alpha \in \{-1, 0, +1\}$ indicates in which side the indicator lays (left, center or right) and ρ is an offset additional to the lane width. The following restrictions apply:

- $\alpha = 0$ for CenterNET.
- $\alpha \in \{-1, +1\}$ for indicators other than CenterNET.
- $\rho = 0$ for lane markings.

The compatibility is assessed based on the average quadratic residual of the lane model $f(\cdot;\sigma)$ evaluated on the coordinates of the road reference. This value is compared against a threshold that is again specific to every

10: **function** RefineModel($l, B_t \dots$)
11: $X_l, Y_l \leftarrow \{\}$ ▷ points that will refine the lane model
12: **for all** $t \in T$ **do**
13: $X_t, Y_t \leftarrow \{\}$ ▷ empty collection of point
14: **for all** $rr \in B_t$ **do**
15: $X, Y \leftarrow$ points of rr
16: **for all** valid α_i, ρ_j **do** ▷ try all the combinations of α and ρ
17: **if** CheckAssociation($l, X, Y, t, \alpha_i, \rho_j$) **then**
18: $X_t \leftarrow \{X_t, X\}$
19: $Y_t \leftarrow \{Y_t, Y + \mathbb{1}\alpha_i(\frac{w}{2} + \rho_j)\}$ ▷ add the translated points
20: **break**
21: **end if**
22: **end for**
23: **end for**
24: merge road references with compatible associations
25: $\sigma_t \leftarrow$ FitModel(X_t, Y_t)
26: add to X_l and Y_l, N_t samples of $f(\cdot;\sigma_t)$ between $[\min X_t, \max X_t]$
27: **end for**
28: **if** $X_l = \emptyset$ **then**
29: discard model l
30: **else**
31: $\sigma \leftarrow$ model parameters of l
32: add to X_l and Y_l, N_{self} samples of $f(\cdot;\sigma)$ between $[\min X_l, \max X_l]$
33: $\sigma \leftarrow$ FitModel(X_l, Y_l)
34: adjust lane width
35: **end if**
36: **end function**

road reference type. The actual value for the threshold must be chosen as a compromise between rejecting non-pertinent measures and adapting to the change of the road shape. If a model doesn't generate any valid association it gets discarded.

To allow for a configurable confidence for each indicator type, the association process is carried out separately for each buffer type. Once all the associations have been determined for a given road reference source, a new model is estimated from their points translated with the associations parameters. This new model is sampled along the x-span of the relative measures, extracting a configurable number of equally spaced samples: these sampled points will be later used to refine the lane model estimate. This process is repeated for every type of road reference that gives positive associations, collecting together the points. To further reduce noise, some samples are collected also from the old model. The new estimate is produced fitting all these points with a new model that will replace the old one.

To improve the stability of the algorithm, the road references from the same source that are associated with the same parameters α and ρ are considered to represent the same physical indicator and are merged together.

The final refinement is made on the estimated lane width. When the model is associated with road references coming from lane markings on both sides, a new estimate of lane width is provided as a difference of the road references' lateral displacement (as computed by algorithm 3.2). To smooth out oscillations due to detection noise, the accepted value is an exponential average of the old width and the new estimation.

3.3.4 Model fitting

Given the coordinate $X = (x_1, \ldots, x_m)^\top \in \mathbb{R}^m$ and $Y = (y_1, \ldots, y_m)^\top \in \mathbb{R}^m$ of a set of points, the model fitting is performed minimizing the mean square error:

$$E(\boldsymbol{\sigma}) = \frac{1}{2} \| Y - f(X; \boldsymbol{\sigma}) \|^2 \tag{3.2}$$

Algorithm 3.5 Model fitting

Input:

$X = (x_1, \ldots, x_m)^\top$ ▷ x coordinates

$Y = (y_1, \ldots, y_m)^\top$ ▷ y coordinates

1: **function** FitModel(X, Y)

2: build $\Phi(X)$ ▷ kernel matrix

3: **return** $\left(\Phi(X)^\top \Phi(X)\right)^{-1} \Phi(X)^\top Y$

4: **end function**

over the model's parameters $\boldsymbol{\sigma} \in \mathbb{R}^n$, where the model function f is applied element-wise on the components x_i of X. The result is the parameters' vector $\boldsymbol{\sigma}^* = (\sigma_1^*, \ldots, \sigma_n^*)^\top$ that satisfy:

$$\boldsymbol{\sigma}^* = \min_{\boldsymbol{\sigma}} E(\boldsymbol{\sigma}) = \min_{\boldsymbol{\sigma}} \frac{1}{2} \| Y - f\,(X; \boldsymbol{\sigma})\,\|^2 \tag{3.3}$$

Exploiting the linearity of f respect to $\boldsymbol{\sigma}$:

$$f\,(x; \boldsymbol{\sigma}) = \sum_{i=1}^{n} \sigma_i \phi_i(x) \tag{3.4}$$

the function evaluation $f\,(X; \boldsymbol{\sigma})$ can be rewritten as:

$$f\,(X; \boldsymbol{\sigma}) = \Phi(X)\boldsymbol{\sigma} \tag{3.5}$$

where the matrix $\Phi(X) \in \mathbb{R}^{m \times n}$ is constructed in the following way:

$$\Phi(X) = \begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \ldots & \phi_n(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \ldots & \phi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_m) & \phi_2(x_m) & \ldots & \phi_n(x_m) \end{pmatrix} \tag{3.6}$$

The problem can be expressed as:

$$\boldsymbol{\sigma}^* = \min_{\boldsymbol{\sigma}} \frac{1}{2} \| Y - \Phi(X)\boldsymbol{\sigma} \|^2 \tag{3.7}$$

from which is evident that $E(\boldsymbol{\sigma})$ is a convex function of $\boldsymbol{\sigma}$. From the convex optimization theory, follows that if a local minimum exists, it is also a global minimum [78]. Therefore $\boldsymbol{\sigma}^*$ must be solution of:

$$\frac{\partial E}{\partial \boldsymbol{\sigma}} = \mathbf{0} \tag{3.8}$$

Expressing the equation as:

$$\frac{\partial}{\partial \boldsymbol{\sigma}} \frac{1}{2} (Y - \Phi(X)\boldsymbol{\sigma})^\top (Y - \Phi(X)\boldsymbol{\sigma}) = \mathbf{0} \tag{3.9}$$

it can be shown that the solution is:

$$\boldsymbol{\sigma}^* = \left(\Phi(X)^\top \Phi(X)\right)^{-1} \Phi(X)^\top Y \tag{3.10}$$

The matrix $\left(\Phi(X)^\top \Phi(X)\right)^{-1} \Phi(X)^\top$ is called Moore-Penrose pseudoinverse of $\Phi(X)$ and can be efficiently computed from the QR factorization [79].

3.3.5 New model creation

The initialization of new lane models is performed through the analysis of lane markings alone. In particular, the lane marking buffer is searched for couple of road reference that are not associated to the same model and are placed at a relative distance compatible with a lane.

If no models are currently being estimated, the road references from lane markings are subject to a pre-association phase that merge together measures that correspond to the same physical marking, as determined by the position and orientation of the road references.

To initialize a new model, the road references are required to have a minimum length and to have been perceived for a minimum time span. These checks help avoid the initialization of spurious models coming from plausible false detections. Another condition is that the model fit procedure gives a bounded residual, meaning that the two markings need to have a similar shape and compatible with possible road models.

Algorithm 3.6 New models creation

Input:

$B_{\text{lm}} = \{rr_1, \ldots, rr_n\}$ ▷ lane markings buffer

L ▷ estimated models

1: **function** SpawnNewModels(B_{lm}, L)
2: **if** $L = \emptyset$ **then** ▷ perform pre-association
3: **for all** $r_i, r_j, i \neq j \in B_{\text{lm}}$ **do**
4: $d_i, o_i \leftarrow$ Displacement(r_i), Orientation(r_i)
5: $d_j, o_j \leftarrow$ Displacement(r_j), Orientation(r_j)
6: **if** $|d_i - d_j| < \text{Th}_d$ **and** $|o_i - o_j| < \text{Th}_o$ **then**
7: merge r_i and r_j
8: **end if**
9: **end for**
10: **end if**
11: **for all** $r_i, r_j, i \neq j \in B_{\text{lm}}$ **do**
12: $d_i, d_j \leftarrow$ Displacement(r_i), Displacement(r_j)
13: **if** $w_{min} < |d_i - d_j| < w_{max}$ **then** ▷ min and max lane width
14: $X, Y \leftarrow$ points of r_i and r_j
15: $l \leftarrow$ FitModel(X, Y)
16: **if** l is feasible **then**
17: **if** $L = \emptyset$ **or** $\nexists \hat{l} \in L |$ **not** Compatible($l, \hat{l}$) **then**
18: $L \leftarrow \{L, l\}$
19: **end if**
20: **end if**
21: **end if**
22: **end for**
23: **end function**

24: **function** COMPATIBLE($l, \hat{l}$) ▷ l new model, $\hat{l}$ estimated model
25: $Y = \{y_1, \ldots, y_n\} \leftarrow$ y-samples of l
26: $Y_{av}, Y_{var} \leftarrow$ average and variance of Y
27: $\hat{Y} = \{\hat{y}_1, \ldots, \hat{y}_n\} \leftarrow$ y-samples of $\hat{l}$
28: $\hat{Y}_{av}, \hat{Y}_{var} \leftarrow$ average and variance of $\hat{Y}$
29: **if** $|Y_{av} - \hat{Y}_{av}| < w_{min}$ **or** $|Y_{var} - \hat{Y}_{var}| >$ Th **then**
30: **return false**
31: **else**
32: **return true**
33: **end if**
34: **end function**

When new models are created, they are checked against some validation criteria like having limited lateral offset, curvature and orientation offset. Moreover, when there other models already exist, these are used to ensure that the new models don't lay in the area of an existing lane and that the shape is consistent withing the same road. The initial width for the new models is set to a default parameter and later refined with lane markings.

3.3.6 Output computation

The final step of the algorithm is the output computation. This output is intended to be used as a navigation reference for the planning stage.

To comply with the planner requirements, the Cartesian coordinates of the lanes' centerline must be augmented with orientation, curvature and width. While the width is directly estimated, the orientation and curvature can be easily recovered from the first and second derivative of the lane model f. Each point $p_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$ has orientation and curvature equal to:

$$\theta_i = \operatorname{atan}\left(f'(x_i)\right) \tag{3.11}$$

Algorithm 3.7 Output computation

Input:

L ▷ estimated models

function ComputeOutput(L)
  $O \leftarrow \{\}$
  for all $l \in L$ **do**
    $\sigma \leftarrow$ model parameters of l
    $P = \left\{ p_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} | y_i = f(x_i; \sigma) \right\}$ ▷ position sampled from the model
    $\Theta = \{\theta_i = \operatorname{atan}(f'(x_i; \sigma))\}$ ▷ orientation of the lane
    $K = \left\{ \left[\frac{f''(x_i;\sigma)}{1+f'(x_i;\sigma)^2} \right]^{\frac{3}{2}} \right\}$ ▷ curvature of the lane
    $O \leftarrow \{O, P, \Theta, K\}$
  end for
  return O
end function

$$\kappa_i = \left[\frac{f''(x_i)}{1 + f'(x_i)^2} \right]^{\frac{3}{2}} \tag{3.12}$$

Based on proximity considerations, the lanes are also ordered and the left-right relationship are reflected in the data structure used to communicate with the planner.

3.4 Results

The road path estimation algorithm has been successfully deployed on DEEVA. It enabled stable and comfortable navigation with reference from sensory input. Quantitative results are presented for two different scenarios, indicated with I and II. These two test case feature different environments with distinct challenges.

The results are presented using a simple 2^{nd} order polynomial as lane model:

$$f(x;\sigma) = \sigma_0 + x\sigma_1 + x^2\sigma_2 \tag{3.13}$$

This model, while being very simple, it's sufficient to represent a variety of situations and shows a good compromise between noise rejection and expressive power.

The ground truth used to provide the results is obtained from the CenterNET annotation laps: it's the recorded position of the car driven in the middle of the lane (see section 2.3.1 for details). For this reason, it should be kept in mind that the accuracy of the ground truth itself is within the limits of human driving. The error is presented as the distance of the estimated centerline from the ground truth and is divided based on the distance of the estimate from the car. Note that the most relevant error is the closest to the car (within the first 10 meters), because it's the one actually used to navigate. The accuracy of the prediction further from the car it's still important to plan the path ahead, but before the vehicle reaches those points they will be iteratively recomputed. To highlight the improvement due to the fusion of multiple source, the error will also be computed using lane markings as the only input.

To present a fair evaluation, in both scenarios the results exclude the roundabouts because they're beyond the limits of both the perception system and the lane model representation.

The following sections presents the details and results for the two scenarios.

3.4.1 Scenario-I

The first presented test case, scenario-I, is a rural road with two lanes for each driving direction, corresponding to the "Parma fiere" area, picture 3.5. Figure 3.6 shows a camera image of the environment in the condition used for testing.

In scenario-I the lane markings are quite visible and part of the course is lined by a guard rail. Due to the relatively high speed of the track, the

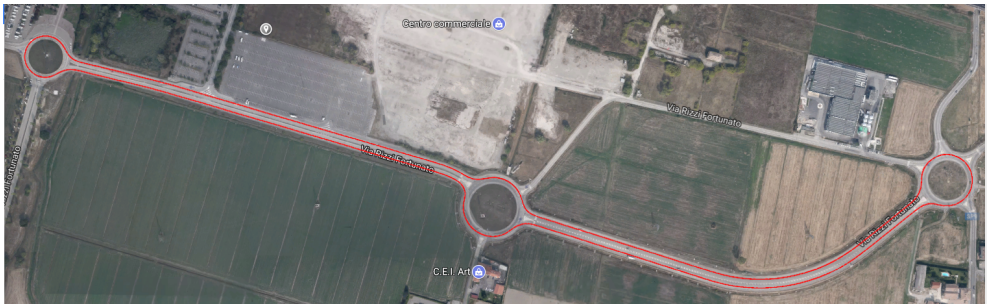


Figure 3.5: Parma fairs, with track highlighted in red.

(a) example of scenario-I

(b) long turn

Figure 3.6: Examples of test scenario I.

algorithm should provide an estimation of, at least, 40m. The major difficulty is represented by a long turn driven in both ways, shown in picture 3.6b. Figure 3.7 shows the track plot where the red areas correspond to roundabouts and have been removed from error computations.

The global error statistics are:

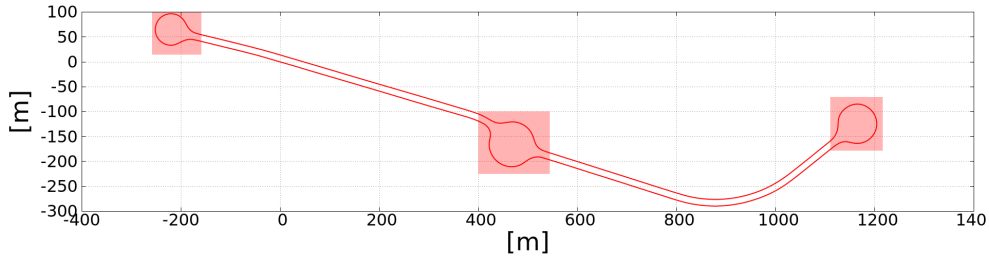


Figure 3.7: Scenario-I track.

Global			
Distance [m]	**Average [m]**	**Max [m]**	**Std [m]**
<10	0.056	0.33	0.05
10-20	0.079	0.48	0.067
20-30	0.12	0.65	0.11
30-40	0.17	1.1	0.19
40-50	0.24	1.8	0.27
50-60	0.26	1.3	0.29

Note that, close to the car, the maximum error is less than 40cm, which is enough to safely navigate the road that has a lane width of 3.4m. Image 3.8

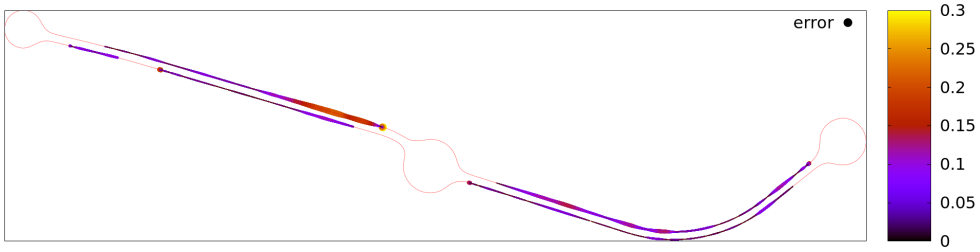


Figure 3.8: Error on the track.

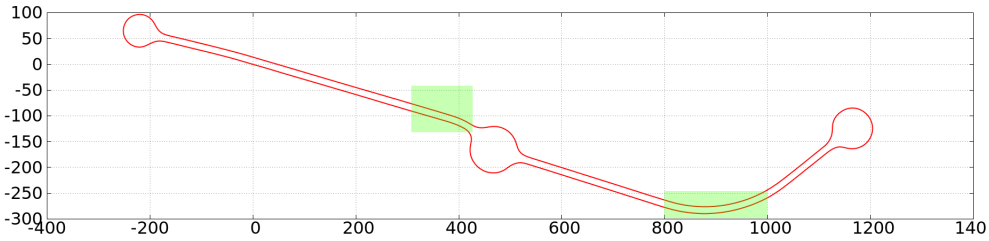
shows the error within the first 10 meters plotted on the track.

To gain intuition of the value added with the fusion of different informations, the next table report the error obtained using only lane markings:

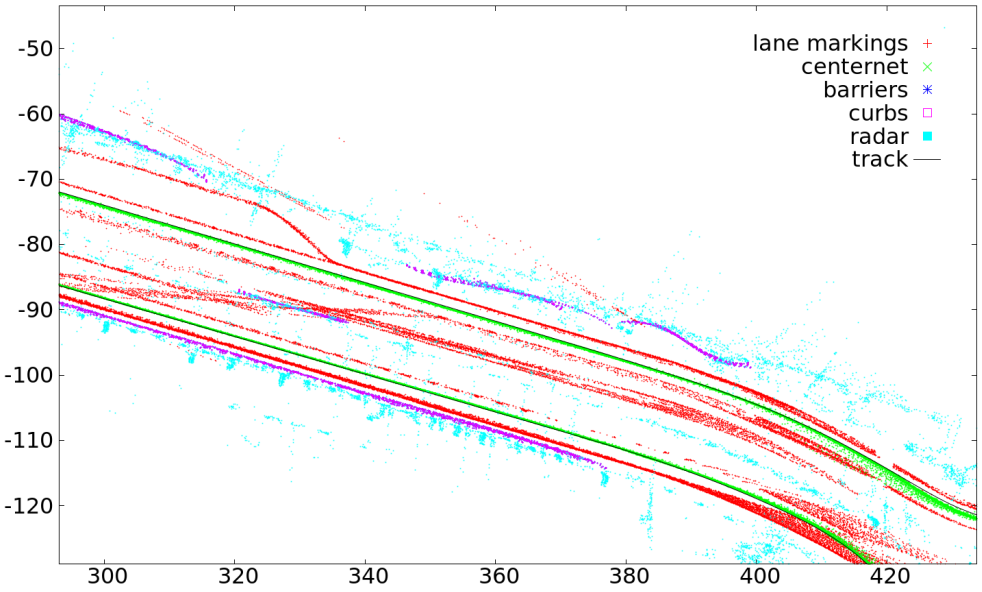
Lane markings only			
Distance [m]	**Average [m]**	**Max [m]**	**Std [m]**
<10	0.16	0.85	0.15
10-20	0.15	0.78	0.15
20-30	0.16	0.85	0.15
30-40	0.17	1.1	0.19
40-50	0.24	1.7	0.27
50-60	0.24	1.3	0.33

Note how, due to the good visibility of lane markings, the average error is very similar. However in the proximity of the vehicle, using other informations improves and stabilizes the estimation, reducing also the maximum error.

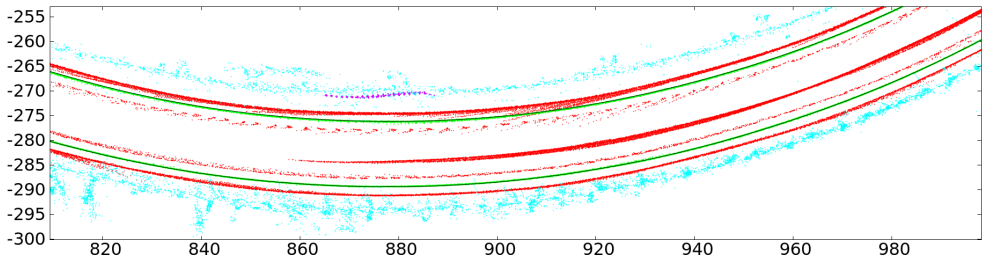
To provide a further hints on the filtering action of the algorithm, in figure 3.9 a detail of the course is plotted with all the raw measures overlaid.



(a) Areas where the raw measurements are shown.



(b) Central roundabout



(c) Long turn

Figure 3.9: Raw measures in scenario-I.

3.4.2 Scenario-II

(a) Cluttered urban environment. (b) Tight turn with little indicators.

Figure 3.10: Test scenario-II: Parmamia.

The second test case, shown in image 3.10, comes from a urban environment with a single lane for driving direction. It corresponds to the "Parmamia" area in the city of Parma, image 3.11. This scenario is much more challenging as it features tight turns and a cluttered environment that induces many false detections. It also pushes to the limit the simple 2^{nd} order polynomial model.

The hardest segment is a narrow road, without lane markings and little of other indicators, that exhibits a turn close to 90° followed by a change in the curvature direction, figure 3.10b corresponding to the green area in plot 3.12. This is the only situation where the algorithm is not able to consistently provide a navigation reference. It would be possible if the parameters were tuned specifically for this case, but the settings are kept constant for all the run.

The global error statistics are:

Figure 3.11: Satellite view of test scenario-II.

Global			
Distance [m]	**Average [m]**	**Max [m]**	**Std [m]**
<10	0.16	1.5	0.13
10-20	0.25	2.3	0.19
20-30	0.46	4.8	0.4
30-40	0.59	4.6	0.37

and image 3.13 shows how the error is distributed in the track. Note how the biggest error correspond to the left turn at the top: right after the turn the model is invalidated because the sensible detections becomes very sparse and, due to model limitations, the algorithm is unable to perform correct associations. This situation is depicted in figure 3.14b.

The errors obtained using only lane markings are the following:

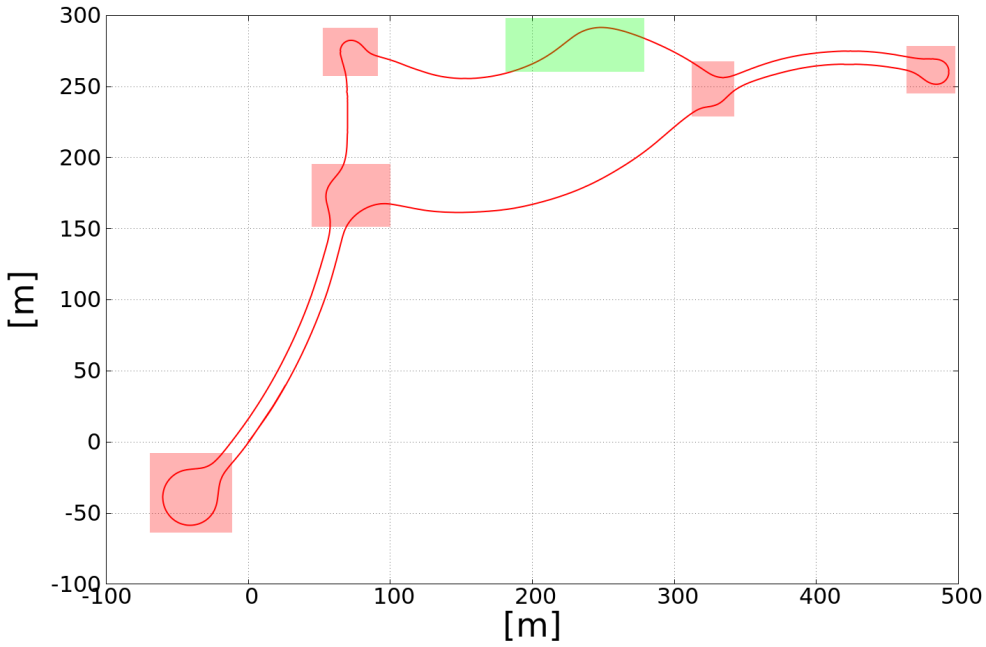


Figure 3.12: Track of test scenario-II. Red areas correspond to roundabouts and the green square indicates the hardest pass of the course.

Lane markings only			
Distance [m]	**Average [m]**	**Max [m]**	**Std [m]**
<10	0.35	7.8	0.46
10-20	0.75	25	1.7
20-30	0.53	6.2	0.8
30-40	0.71	6.1	1.1

It's evident that the fusion of multiple indicators improves the results and, especially, increases the overall robustness as indicated by the difference in the maximum error.

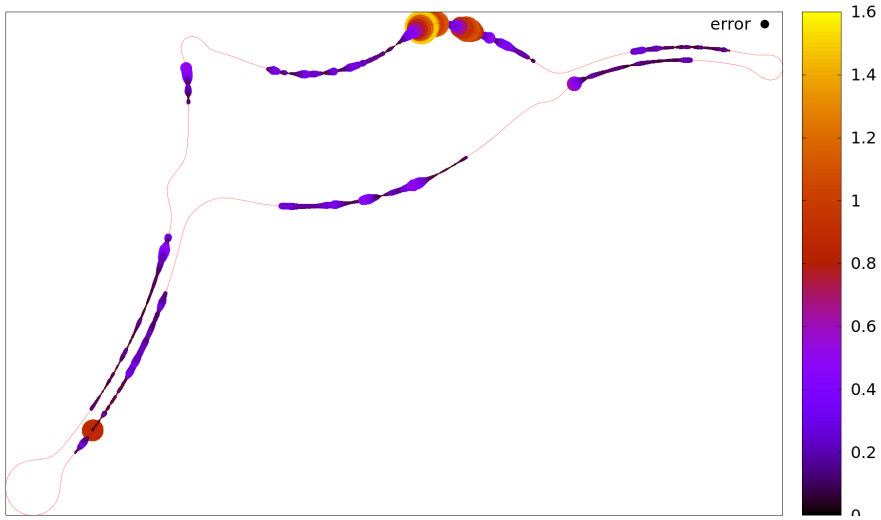
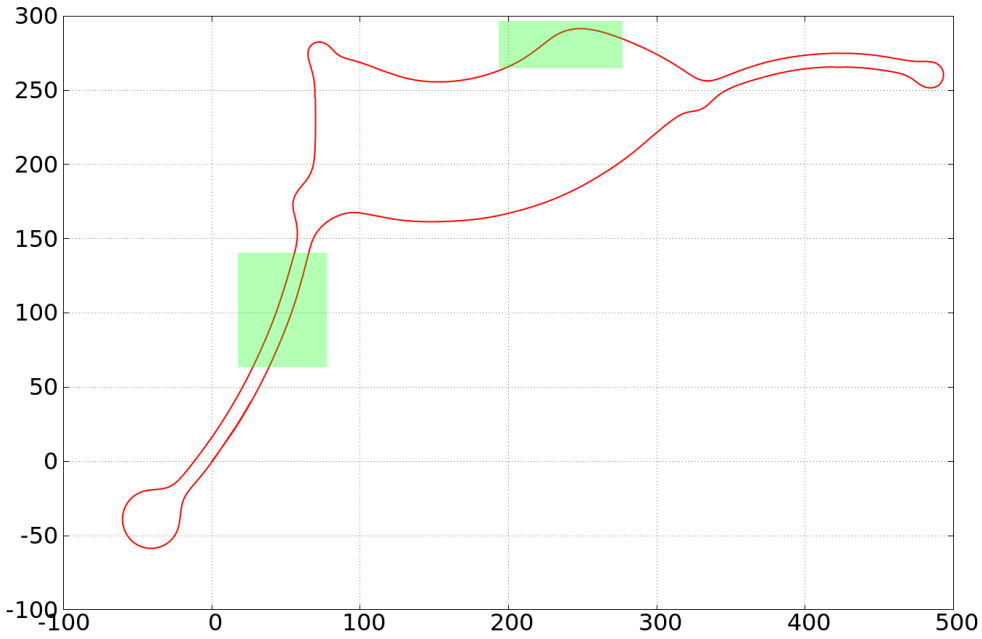
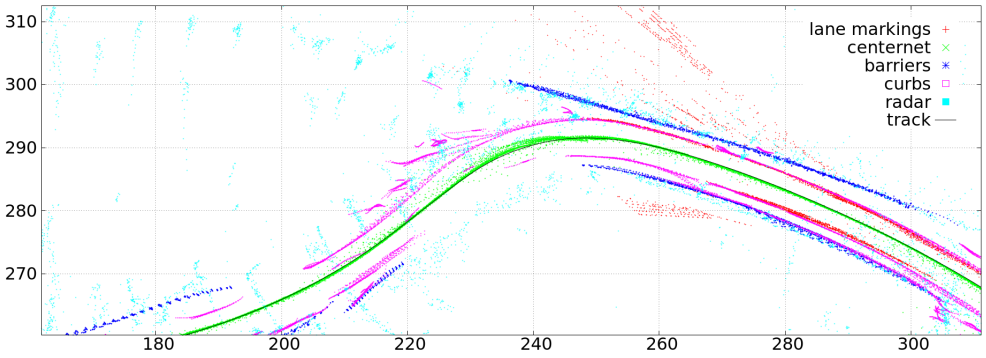


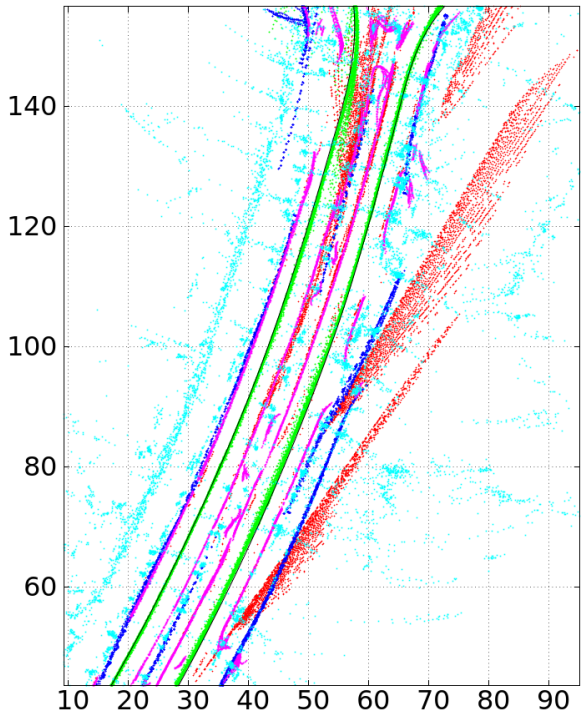
Figure 3.13: Distribution of scenario-II errors on the map.



(a) Areas where the raw measurements are shown.



(b) Left turn on top.



(c) Bottom straight road.

Figure 3.14: Scenario-I raw measures.

3.5 Conclusions and future development

The previous section introduced a fusion algorithm for road-path estimation from multiple road indicators. The tests conducted on different scenarios point out the ability of the proposed approach to leverage the different information sources and provide a stable measure of the road path. It shows consistent improvement over the estimation from lane markings only in terms of accuracy and stability of the measure. The algorithm was also successfully employed on DEEVA to perform autonomous navigation from sensory reference.

There are however some limitations that must be highlighted.

First, constraining the road model to be a *function* in body coordinate can be too limiting, especially in urban environment. In particular, roundabouts cannot be represented because the lane has multiple y values for the same x. This limitations could be removed switching to a bi-dimensional approach, like potential fields, but this would greatly impair the ability to ignore false detections. Overall, the benefits of imposing a model seems to overcome the shortcomings. Another problem with using a fixed model is that it can fail to let the algorithm adapt to the changing condition of the road. If we had clues about the approximate shape of the road, coming for example from ADAS maps, the algorithm could adapt the model to the actual conditions. For example, knowing that a road is straight, the algorithm could use first grade polynomial further improving the noise rejection.

The other major limitation is the need of manual parameter tuning. The procedure's outcome depends on many configurations parameters that need to be adjusted for the different environments. To overcome this problem, the algorithm could adjust the additional offsets ρ observing the distance of other road indicators with respect to the associated lane markings. Such process would introduce more variance in the estimation, but could be able to adapt the parameters to the changes in the environment.

Chapter 4

Planning: Local Trajectory Computation

4.1 State of the art

The problem of planning arises in any case where there is an agent that needs to achieve some goal through actions. This problem is ubiquitous in robotics and it is of central importance for autonomous vehicles.

The formulations of the planning problem are as various as its possible applications. In the context of autonomous driving, in the most general terms, *planning* refers to the process through which the vehicle exhibits intelligent behavior given an understanding of the surrounding environment. This general definition is usually broken down into 3 levels of decreasing abstraction:

Route planning: this is the highest level of abstraction in the planning process. The goal is to select a route through the road network that would take the vehicle from the current position to the requested destination. The road network is normally represented as a directed graph and the problem reduces to graph path searching. However, due to the potential large (continental) scale of the graph, classic approaches like A* and Dijkstra [80, 81] might be unfeasible and specialized large-scale algo-

rithm must be used [82]. This functionality is analogous to that of a GPS navigator and is required only to achieve autonomy of SAE level 3 or higher.

Behavioral planning: once a sequence or roads has been set, the vehicle needs to navigate those roads and act according to the road infrastructure and other road participants. Defining a sequence of driving behavior to safely navigate along the route is the task of the behavioral planner. This refers to taking over the tactical aspects of driving, for example deciding when to change lane, stop at a stop line and yield at intersections. The behavioral planning problem has a discrete nature and methods based on *finite state machines* have been the dominant approach for years [83]. The need to model uncertainty and more complex interactions, led to the use of probabilistic planning formalisms based on Markov Decision Processes and their Partially-Observable variants [84, 85].

Local trajectory planning: this is the planning level with the lowest abstraction, that deals directly with the car movement. After choosing a driving behavior, a motion planner is responsible for finding a reference trajectory that implements the required behavior. The term *trajectory* is used to indicate a feasible series of states for the system (position and orientation for the car), with the additional information of the temporal instants they should be visited.There are several approaches to the solution of the motion planning problem.

In my work, I focused on the local trajectory planning. The problem can be formalized in the following way. Let $\mathcal{X}$ be the configuration manifold and $\mathcal{X}_{obs} \subseteq \mathcal{X}$ the set of configurations that results in the system being in collision with obstacles. Defining $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}_{obs}$ the set of free configurations and $\mathcal{X}_{goal}$ the set of target configurations, such that $\mathcal{X}_{goal} \cap \mathcal{X}_{free} \neq \emptyset$, the trajectory planning can be stated as finding a curve, function of time, $\pi : [0, T] \to \mathcal{X}$ such

that:

$$\pi(0) = \mathrm{z}_{init}$$
$$\pi(T) \in \mathcal{X}_{goal}$$

subject to:

$$\pi(t) \in \mathcal{X}_{free} \quad ,\forall t \in [0,T]$$
$$D(\pi, \pi', \pi'', \dots) \quad ,\forall t \in [0,T]$$

Where $D(\cdot)$ indicates a set of possible additional constrains that must be satisfied.

In the context of autonomous cars, $\mathcal{X}$ is usually the pose and speed of the vehicle so that $\mathrm{z} = (x, y, \theta, v)^\top \in \mathcal{X} = \mathbb{R}^2 \times \mathbb{S}^1 \times \mathbb{R}$ specifies the 2D position, orientation and speed of the vehicle. The car model is also subject to differential constraints that can be derived from the equivalent bicycle model [86], shown in figure 4.1. Assuming zero side-slip of the wheel, the kinematic equations

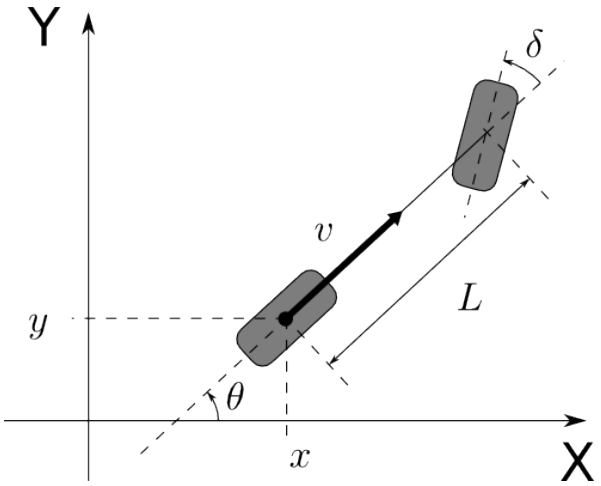


Figure 4.1: Equivalent bicycle model for car-like kinematics.

are:

$$\begin{cases} \dot{x} &= v\cos(\theta) \\ \dot{y} &= v\sin(\theta) \\ \dot{v} &= a(t) \\ \dot{\theta} &= \frac{v}{L}\tan\delta(t) \end{cases} \tag{4.1}$$

where δ is the steering angle, L is the car wheel-base and a the car's longitudinal acceleration. The coordinates (x, y) are the one of the rear axle's center, which is the reference point on the car used to derive the equations. The behavior of the system is determined by the control functions $\delta(t)$ and $a(t)$ that are subject to the bounds constraints:

$$\begin{aligned} \delta_{\min} \leq \delta(t) \leq \delta_{\max} \quad &, \forall t \\ a_{\min} \leq a(t) \leq a_{\max} \quad &, \forall t \end{aligned} \tag{4.2}$$

that represent the intrinsics limits of the vehicle of having a minimum turning radius and limited acceleration/braking power. For a trajectory π to be feasible, it must satisfy (4.1) and (4.2) so they must be included in $D(\cdot)$.

The problem statement is usually reformulated to frame it in the context of the solution method applied, but the basics requirements stay the same: the trajectory must be feasible, free from collisions and reach some kind of goal. In the context of autonomous driving, there are three main approach families to the solution of this problem: lattice search, RRT and variational methods.

Lattice approaches reduce the planning problem to a graph search problem. The graph is generated from a set of proper motion primitives, also called generators, and exhibits a lattice structure meaning that each vertex can be expressed as a linear combination of the graph's generators. The lattice graph shows a regular structure, see image 4.2 for a comparison between lattice and non-lattice graphs taken from [87]. Choosing the appropriate motion primitives, this approach can easily handle dynamic-constrained system [88] and a trajectory can be generated including time in the space of the graph [89]. Removing edges that intersects obstacles, the planning is then performed searching a path in the graph that reaches the goal region while optimizing some cost.

RRT, which stands for *Rapidly exploring Random Trees* [90], is another family of approaches that requires the search over a graph, but the graph generation process is profoundly different. The "rapidly exploring" property comes from the fact that the vertices of the graph are sampled from a distribution that covers all the available space. When a new vertex is drawn

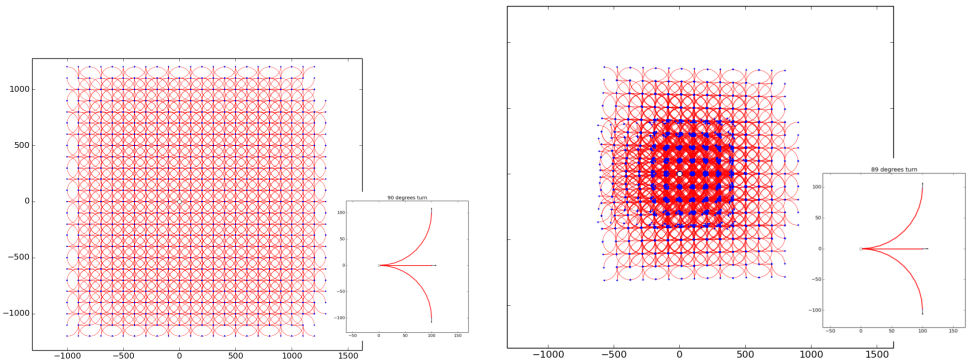


Figure 4.2: Comparison of the space coverage of lattice graph (left) and non-lattice graph (right).

from the appropriate distribution, the graph is expanded including an additional vertex that connects the new one with the closest edge. The procedure that connect the vertexes can account for the system dynamic and make sure that the path is collision-free. While the original algorithm has been shown to converge, almost surely, to a sub-optimal solution, the variant RRT* [91] is probabilistically optimal given enough computation time.

A totally different approach is the one of (direct) variational methods, that frame the problem of planning in the general framework of non-linear continuous optimization. So the planning problem is expressed as:

$$
\begin{aligned}
\underset{\pi \in \mathcal{X} \times \mathcal{T}}{\text{argmin}} \quad & J(\pi) \\
\text{subj. to:} \quad & \pi(t) \in \mathcal{X}_{free} && ,\forall t \in [0,T] \\
& \pi(0) = \mathrm{z}_{init} \\
& \pi(T) \in \mathcal{X}_{goal} \\
& f(\pi, \pi', \pi'', \dots) = 0 && ,\forall t \in [0,T] \\
& g(\pi, \pi', \pi'', \dots) \leq 0 && ,\forall t \in [0,T]
\end{aligned}
$$

For a properly designed J. To leverage the existing non-linear optimization methods, the infinite-dimensional trajectory space is projected on a finite-dimensional vector space. The planning is then performed optimizing J over

this vector space, mapping vectors to possible trajectories [92, 93].The system dynamic is handled as equality constraints of the optimization problem.

These approaches are fundamentally different and further considerations will be provided in the next section.

4.2 Proposed approach

All the approaches presented have been employed in the field of autonomous driving, but they have very different properties and they're not all suitable to be used in the context of this research.

Lattice search, while being very successful with mobile robots, can pose substantial difficulties in the case of self-driving car. The main problem is the choice of the motion primitives used to generate the graph. With a very structured environment, like a urban road, finding primitives that both generate a lattice and adapt to the widely varying road shape can be difficult. Even discarding the requirement of the graph being a lattice, the size is exponential in the number of motion primitives (each node can be expanded with any of them) so choosing a large number of primitives can lead to a huge graph, and thus long computation time, while selecting too few can result in rough final trajectories.

Also RRT have some shortcomings: the random sampling might results in wasted computation time to explore regions of little interest. Even for RRT* a large number of iterations may be required to find a trajectory free from unwanted oscillations. Moreover, the extensive computation required to build the whole tree offers limited benefits in a highly dynamic, unpredictable environment and alternate version designed to explicitly deal with replanning [94] have a run time of several tens of seconds. Overall, their random and iterative nature makes them mostly unfit for real-time applications.

On the other hand, variational methods are widely know to quickly converge to local minima of the cost function. While this has been seen mostly as a problem, we must recognize that, with a properly designed cost function,

the fact that the algorithm doesn't provide the true global optimum is of little practical interest, as long as the solution is safe and comfortable. Moreover, the variational methods have the tremendous advantage of allowing the optimization of an arbitrary cost function J. This flexibility helps overcome what I think is one of the biggest limitations of classic approaches: the definition of $\mathcal{X}_{goal}$. While some scenarios offer a natural concept of $\mathcal{X}_{goal}$, i.e. the final pose for a parking maneuver, there are cases where these intermediate goals are not readily available. As an example, the navigation of a lane has the final goal of reaching the end of it (maybe the next junction) but that might be beyond the planning horizon when the car is just at the beginning of the road. Choosing intermediate goal regions along the way is less straightforward than it seems:

- The goal region must be actually accessible, otherwise the algorithm might run forever. Even in case of safeguard to prevent infinite loops, there must be a criterion to select *the* best trajectory among those that fail to reach the goal.

- Having a goal region that is too wide, to prevent unaccessible regions, might instead result in a premature algorithm termination for reaching a goal that is too close. Preventing this behavior means having, again, a criterion to establish which final point / solution trajectory is better among those that reach the goal zone.

These difficulties arise because the problem actually lacks the concept of intermediate goal. Beyond staying within the lane boundaries, we don't want to constraint the car to be in a certain region at a certain time, we just want it to progress safely along the road toward the true final goal. This concept of *progress* fits nicely in the optimization framework.

Because of their efficiency and flexibility, I decided to adopt a variational approach to the solution of the trajectory planning problem. The outline of the developed method is given next, with details in the following sections.

The planning problem is known to incur in what's known as "the curse of dimensionality" [95], that basically imply that the problem *difficulty* is exponential in the dimensionality of the solution space. Since the trajectory belong to the Cartesian product of the configuration space and time span, $\mathcal{X} \times \mathcal{T}$, it implies that solving for a trajectory is strictly harder than finding a path and, subsequently, the speed to traverse it. This would amount to finding two functions $\pi : [0,1] \to \mathcal{X}$ and $\sigma : [0,T] \to [0,1]$ that combined together are equivalent to a trajectory. With little reformulation, the kinematic car model admits a clean separation between the path, determined by the curvature function, and its travel speed, which is the longitudinal car velocity. To improve the computational efficiency, the trajectory planning problem is hence split in the path planning and speed tuning problems. This technique, called "speed tuning method", is a known trick to reduce the computational complexity of planning problems [96].

The proposed approach exploits the speed tuning method and split the problem in path and speed planning. Path planning is solved with a variational approach, while the speed is computed with reactive policies.

To properly define the scope of the developed trajectory planner, I restrict my focus to the problem of navigating a reference path with given boundaries, obstacles and speed limits. This focus is not excessively restrictive since such trajectory planner can be used to perform lane keeping, lane change and junction negotiation and thus covers most of the driving scenarios, given the proper behavioral planner and reference path.

The navigation reference is assumed to be provided as a sorted series of waypoints that have the following information:

- 2D Cartesian coordinate of the center.
- Orientation.
- Width.
- Curvature.

- Speed limit.

Of these information, the coordinate of the center, orientation and width are used to perform path planning, while the curvature and the speed limit are relevant to choose the proper speed for the planned path. When, during lane change, two or more lanes are available they will be represented as separate navigation references with adjacency informations.

For what concern the obstacles, the separate solution for path and speed forces a distinction on their type. There are obstacles that must be avoided with evasion maneuvers and thus influence the shape of the path. These are mostly static obstacles that are found within the road boundaries (parked cars and general small obstacles) and will be called *raw obstacles*. There are, however, obstacles that must be avoided without changing the shape of the driven path, but adjusting the car speed accordingly. These are typically the other road participants, vehicles and crossing pedestrians, and will be called *road obstacles*.

With the introduced distinctions, the following sections will explain in detail the planning strategies for both path and speed.

4.3 Path planning

For the sake of efficiency, the trajectory planning problem is solved as path planning followed by speed planning. To perform this separation, the derivative of the state variables $(x, y, \theta)^\top \in \mathbb{R}^2 \times \mathbb{S}^1$ are taken with respect to the traveled space:

$$x' = \frac{dx}{ds} = \frac{dx}{dt}\frac{dt}{ds} = \dot{x}\frac{1}{v} \tag{4.3}$$

Thus the system (4.1) becomes:

$$\begin{cases} x' &= \cos(\theta) \\ y' &= \sin(\theta) \\ \theta' &= \frac{\tan(\delta)}{L} \end{cases} \tag{4.4}$$

where the equation relative to the dynamic of v has been discarded. Introducing the vehicle curvature $\kappa = \tan(\delta)/L$, the final dynamic is:

$$\begin{cases} x' &= \cos(\theta) \\ y' &= \sin(\theta) \\ \theta' &= \kappa \end{cases} \tag{4.5}$$

with the constraint:

$$\kappa_{\min} \leq \kappa \leq \kappa_{\max} \tag{4.6}$$

Since the integration variable is the traveled space s, also the control κ is a function of s.

The manifold $\mathbb{R}^2 \times \mathbb{S}^1$, obtained from the Cartesian product of $\mathbb{R}^2$ with the unit-circle $\mathbb{S}^1$, is called *Special Euclidean* group of order 2 and will be indicated with SE(2) [97].

The inputs to the path planning process are:

- The initial state for the car z_{init}.
- The navigation reference R, as a series of waypoints with position, width and orientation.
- The raw obstacles O_r.

Equations (4.5) constrain the evolution of the path in the configuration space SE(2) and hence, given the initial state z_{init}, the path is completely determined by the curvature function $\kappa(s)$ and the total traveled space s_f. For this reason, the projection of the path on a finite-state space ca be easily done finding a parametrization of the curvature function κ. Exploiting the curvature parametrization, the proposed variational approach finds the optimal path P minimizing a cost function $J(P, \kappa; R, O_r)$ over the curvature parameters space, as shown in algorithm 4.1.

The next sections detail the ingredients of the path planning procedure.

Algorithm 4.1 Path planning procedure

Input:

z_{init}	▷ initial state
R	▷ navigation reference
O_r	▷ raw obstacles

Output: best path P^*

1: $p \leftarrow$ initial guess p_0
2: **while** not termination condition **do**
3: $\quad \kappa \leftarrow$ curvature from parameters p
4: $\quad P \leftarrow$ path determined by κ and z_{init}
5: $\quad c \leftarrow J(P, \kappa; R, O_r)$
6: $\quad p \leftarrow$ optimization step with p and c
7: **end while**
8: **return** the path from the best parameters p^*

4.3.1 Curvature parametrization and path computation

Choosing a parametrization basically means constraining the curvature function to belong to a certain functions family. There are infinite possible families, thus some criteria are needed to drive the choice. From the analysis of the problem, the following requirements can be derived:

1. The parametrization should be, relatively, low dimensional to allow a fast optimization.
2. The total length of the path should be derived from the parameters to allow adaptive path lengths.
3. The parametrization should have enough expressive power to allow both smooth driving and sharp avoidance maneuvers.
4. It should be easy to enforce the limited-curvature constraint (4.6).

As an example, [92] utilizes a 5-dimensional parameters vector to represent a 3$^{\text{rd}}$ degree polynomial with the final length. While this representation meets

some of the requirements, it also has some problems. Besides the poor numeric conditioning of polynomials coefficient and the limited expressive power (for example constant radius turns are not part of the family), this representation has the big disadvantage of not having a natural way to enforce the limited curvature constraint. The authors, in fact, were forced to impose the approximate proxy condition:

$$\int_0^{s_f} \kappa(s)^{2\eta} ds < \kappa_{\max}^{2\eta} \tag{4.7}$$

that for $\eta \in \mathbb{N}, \eta \gg 1$ is somehow related to the maximum curvature.

To overcome these limitations, I propose to choose the curvature function to be piecewise linear, such that the parameters represent the control points, equally spaced in its total length. The parameters vector $(\kappa_0, \kappa_1, \ldots, \kappa_{N-1}, s_f)^\top \in \mathbb{R}^{N+1}$ represents a sequence of $N-1$ linear segments $\{(\kappa_0, \kappa_1), (\kappa_1, \kappa_2), \ldots, (\kappa_{N-2}, \kappa_{N-1})\}$ each of length $s_f/(N-1)$. An example for $N = 6$ is provided in figure 4.3.

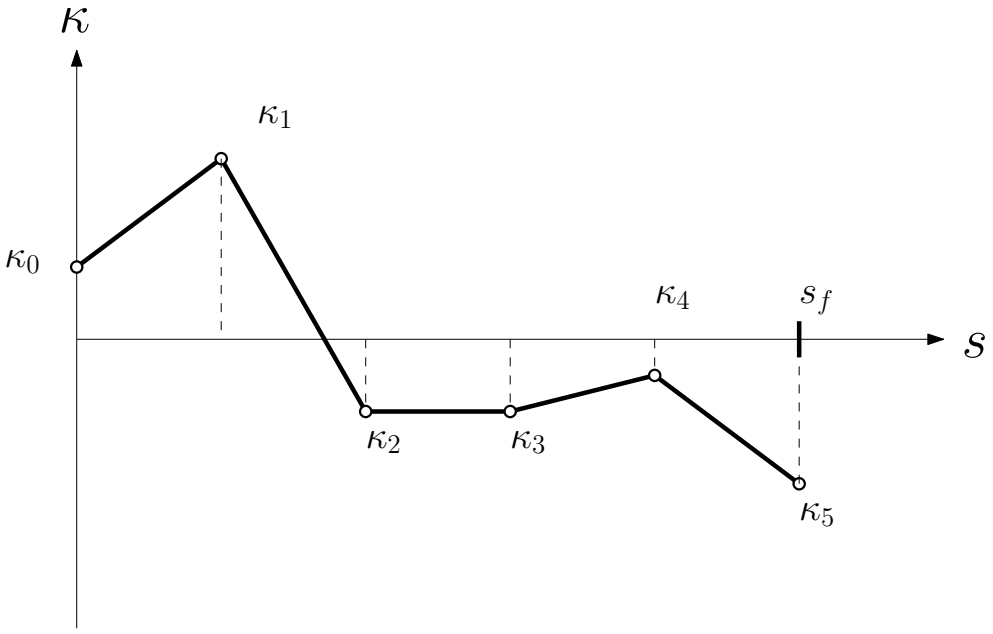


Figure 4.3: Example of curvature profile.

In addition to being arbitrarily low-dimensional and having a variable length, this representation format has the following advantages:

- The limited curvature constraint is trivial to apply since the values κ_i are the local minima/maxima of the function.

- Not only it can represents a broad range of shapes, but the "manoeuvrability" of the path is inversely proportional to its length and thus fits nicely with both long smooth turns and short sharp evasion paths.

A drawback of this format is that the curvature function $\kappa(s)$ has only a piecewise definition:

$$\kappa(s) = \begin{cases} \kappa_i + (\kappa_{i+1} - \kappa_i)\frac{(N-1)s - i\, s_f}{s_f} & ,\text{if } \exists i \mid i \leq \frac{s(N-1)}{s_f} < i+1 \\ \text{undefined} & ,\text{otherwise} \end{cases} \tag{4.8}$$

so a nice derivative expression is not readily available.

The kinematic car model (4.5) defines an ordinary differential equations (ODE) system. Fixing the initial state z_{init} and the curvature function $\kappa(s)$, specifies an initial-value problem and a wide literature exists on the solution of such problem. Since the performance are of utmost importance, error-control solution methods are appealing since they can adjust the solution-step according to the approximation error. The solution algorithm of choice is the Dormand–Prince method of order 5 (DOPRI5) from the family of Runge–Kutta with error control [98]. This algorithm is applied to integrate the system (4.5) from the initial condition z_{init} to the limit value s_f for the integration variable. The intermediate states $\{\mathrm{z}(s) \in SE(2) | s \in [0, s_f]\}$ are the samples of the final path P. While extracting a fixed number of states in $[0, s_f]$ could be computationally advantageous, for long paths this would mean having samples of the path that are spatially very sparse. This sparsity could pose serious threat to safety since states might jump over obstacles and even road boundaries. To avoid such dangerous jumps, the states are extracted with a constant

step Δs, so the path:

$$P = \left\{ \mathrm{z}_i = \begin{pmatrix} x_i \\ y_i \\ \theta_i \end{pmatrix} = \mathrm{z}(i\Delta s) \mid 0 \leq i\Delta s \leq s_f \right\} \tag{4.9}$$

has a variable number of elements that depends on the overall length s_f. Figures 4.4 and 4.5 show examples of curvature functions and resulting paths.

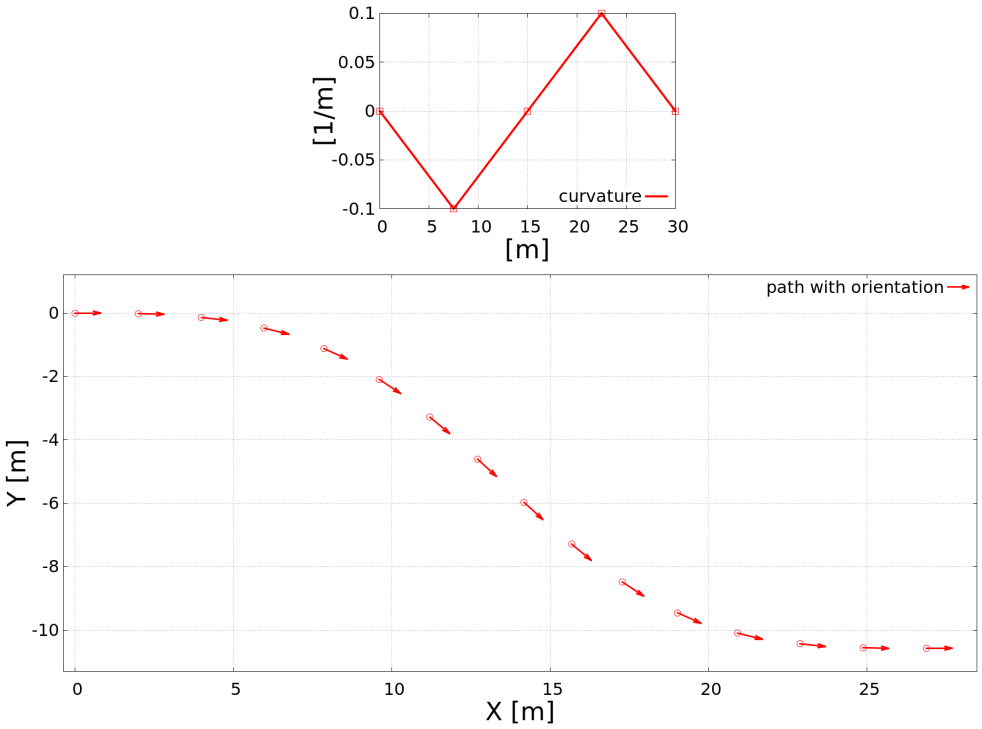


Figure 4.4: Curvature profile and relative path.

This section detailed the parametrization of the curvature function and the procedure to map this curvature to a path, given an initial state.

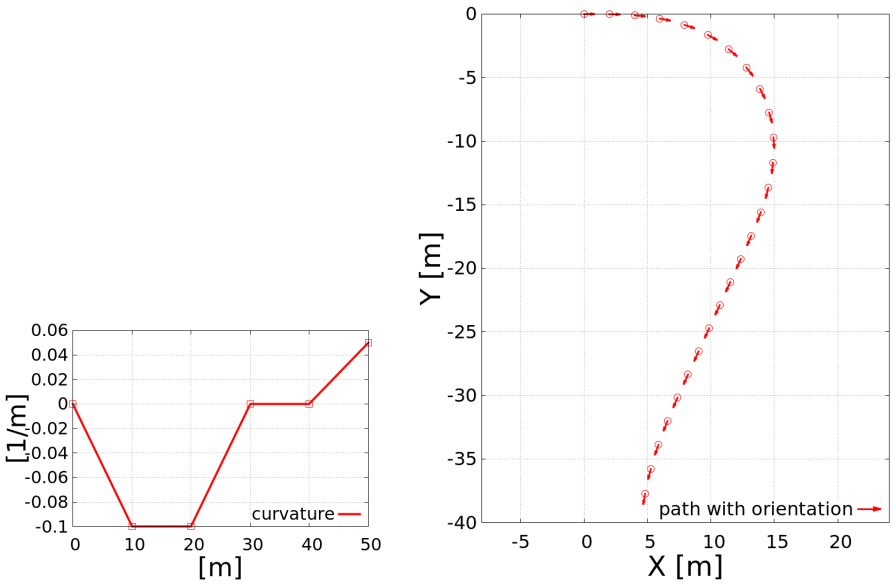


Figure 4.5: Another example of curvature and resulting path.

4.3.2 Cost function and optimization

The great flexibility of the variational methods resides in the design of the cost function J that maps a path P to a scalar cost c. This function should be shaped to account for all the requirements on the final path:

1. It must stays within the road boundaries.
2. It must avoid collisions with static obstacles.
3. It must be comfortable.
4. It must progress along the road.

As it's often the case, the requirements can be conflicting and some sort of importance weights should be assigned to them. Of course safety concerns should be primary, so requirements 1 and 2 are of higher importance than 3 and 4.

To further guide the choice of J, there's another important consideration about the computational performances. Given that the computation complexity is inevitably dependent on the number of elements in P, it should be avoided introducing a further dependence on the (unknown) number of obstacles. While the size of P can be limited constraining the total length s_f, the number of obstacles comes from the environment and there's no control on their maximum number. For this reason, the common approach of assigning a cost inversely proportional to the car-obstacles distance, as measured from the minimum vertex-vertex distance, introduces a potentially large computation burden on each evaluation of J. A convenient solution that permits the seamless integration of boundaries and obstacles and that doesn't introduce explicit computation dependence on the number of obstacles is the use of a *potential function*. Assigning high cost to boundaries and obstacles forces the path to stay within the road and avoid collisions.

Since the optimization is likely to converge to a local minimum, J must be designed to admit minima that correspond to acceptable paths. An information that helps the convergence to good paths is the orientation error between the car and the track. The dynamic of θ in the ODE is basically an integration of $\kappa(s)$, so assigning an error on the orientation provides a strong guidance toward curvature profiles that keep the car orientated like the road, which is often the right choice.

To address the comfort issue, the path should avoid unnecessary oscillations and abrupt sharp turns. This factor can be evaluated directly from the $\kappa(s)$ function.

Finally, J should encourage the path to progress along the road, when possible. This progress can be measured tracking the road span covered by the path: the higher the difference between the first and last points on the road, the higher the bonus for advancing along the reference.

This concludes the description of the necessary components of the cost function J. The details for each contribution will be presented in the following sections.

Potential function

The potential function $\Phi : \mathbb{R}^2 \to \mathbb{R}$ assigns a scalar value to every point in the space that represents the *cost* of traversing that area. This function can embed any information that translates to zones that should be preferred or avoided, so road boundaries and obstacles are easily embedded in the potential function.

Since the potential doesn't depends on the path being evaluated, but only on the navigation reference R and the raw obstacles O_r, the values of Φ can be computed in advance to provide a fast access during the optimization process. To precompute the function's values, the space must be discretized in some way and the potential evaluated on the selected locations. While using a square Cartesian grid would be straightforward, this would result in computation and memory being wasted for zones of no real interest since the width of this grid should be large enough to include turns in the navigation reference. Choosing a discretization that is memory-efficient requires to restrict the attention on the most relevant space portion: the navigation reference. To achieve an efficient sampling of Φ, I defined the potential on the coordinate system generated by the curvilinear abscissa along the reference R and the orthogonal distance from the identified point along the curve. These coordinates will be called

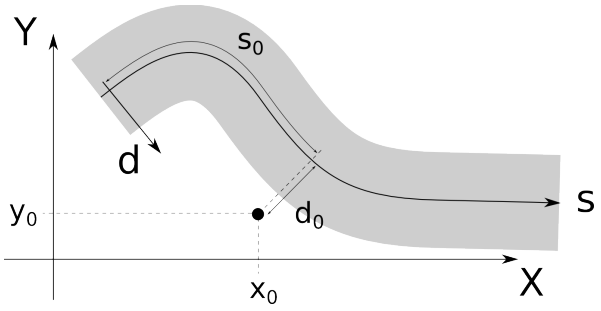


Figure 4.6: Cartesian and road coordinates for the same point.

road coordinates and indicated with s (curvilinear abscissa) and d (lateral orthogonal distance). Figure 4.6 shows the relationship between the road and

Cartesian coordinates. Sampling Φ over a road coordinates grid provides an efficient coverage of all the relevant space for the navigation task.

To utilize such potential function, conversion between Cartesian and road coordinates is needed. The conversion equations depend on the navigation reference $R = \{\mathrm{w}_1, \mathrm{w}_2, \ldots, \mathrm{w}_n\}$ as the road coordinates unfold with the series of waypoint. To provide a fast and efficient conversion, a linear approximation of R will be considered, effectively discarding the orientation and curvature information. Each waypoint w_j is assumed to have Cartesian coordinates $(x_j, y_j)^\top$ and curvilinear abscissa s_j. Since the conversion from Cartesian to road is not generally unique, the following assumptions will hold:

1. A Cartesian point is converted to the road coordinates that results in the lowest orthogonal displacement.

2. In case of multiple lowest orthogonal displacements, the point is projected on the lowest curvilinear abscissa.

This amount to force the conversion procedure to return, among the possible results, the least in lexicographic order over d and s. The pathological situations resolved by the introduced assumptions are shown in figure 4.7. Given

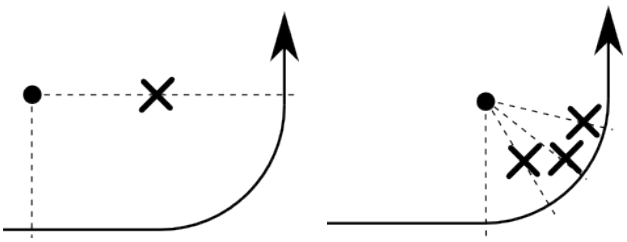


Figure 4.7: Ambiguous projections, wrongs marked with 'X'.

a Cartesian point ${}^{c}p = (x_p, y_p)^\top$, the closest segment $(\mathrm{w}_i, \mathrm{w}_{i+1})$ is identified computing the point-segment distance and using assumptions 1-2. Introducing:

$$\sigma_i = \mathrm{w}_{i+1} - \mathrm{w}_i = \begin{pmatrix} x_{i+1} - x_i \\ y_{i+1} - y_i \end{pmatrix} = \begin{pmatrix} \Delta x_i \\ \Delta y_i \end{pmatrix} \tag{4.10}$$

$$\sigma_i^{\perp} = \begin{pmatrix} \Delta y_i \\ -\Delta x_i \end{pmatrix} \tag{4.11}$$

The road coordinates of ${}^r p = (s_p, d_p)^\top$ are obtained through orthogonal projections on σ_i and $\sigma_i^{\perp}$:

$$s_p = s_i + \langle {}^c p - \mathrm{W}_i, \frac{\sigma_i}{\|\sigma_i\|} \rangle \tag{4.12}$$

$$d_p = \langle {}^c p - \mathrm{W}_i, \frac{\sigma_i^{\perp}}{\|\sigma_i\|} \rangle \tag{4.13}$$

where $\langle \cdot, \cdot \rangle$ denotes the scalar product and s_i is the curvilinear abscissa associated to W_i. The graphical representation of this process is depicted in figure 4.8.

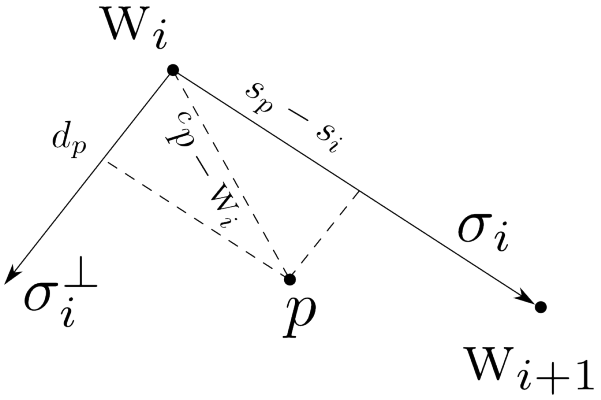


Figure 4.8: The coordinate (d_p, s_p) obtained through projection on σ_i and $\sigma_i^{\perp}$.

One remark is needed about the evaluation of the potential function. Since the car is not a punctual entity its encumbrance must be accounted for. Instead of evaluating Φ at the corners of a box approximating the car shape, to reduce the number of required evaluations, the width of the car w_c is considered padding boundaries and obstacle with half its value. With a slightly larger car width, this approximation is *safe* for small angles between the car

and the road, which is a reasonable assumption. The length of the car l_c , conversely, is accounted using two evaluation points: one centered in the vehicle's coordinates and another corresponding to the tip of the vehicle, figure 4.9. Of

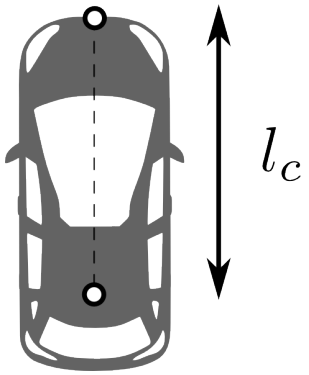


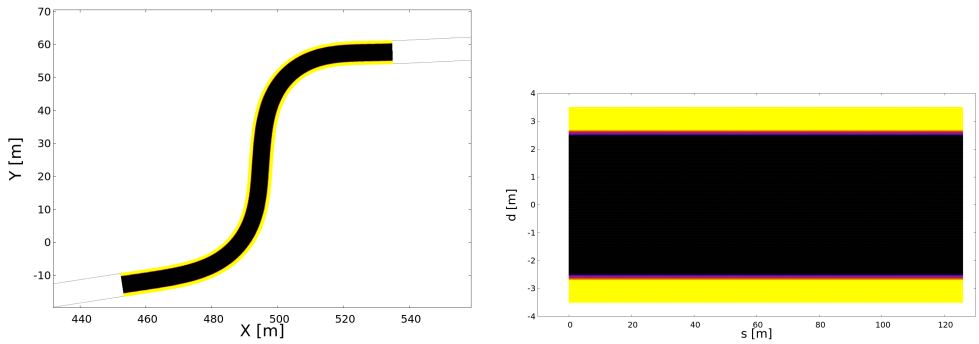
Figure 4.9: Evaluation points.

these two values, the maximum is assumed to be the potential value for the given car state.

With the conversion equations established, the process of embedding information on the potential function is described next. Focusing on the *sampling* of the function Φ, the potential is treated as being an image whose pixel $\Phi_{i,j} = \Phi(i\Delta s, j\Delta d)$ represents the samples' value. The sampling steps Δs and Δd establish how the samples are distributed in the (s, d) space. In the following, metric measures are implicitly converted to pixel using Δs and Δd. With this perspective, the embedding of information in Φ is called *potential painting*.

Using road coordinates, boundaries have a very natural representation: for every value of s, they correspond to $d = \pm w/2$, where w is the reference width. Given the value assigned to boundaries p_B, the potential image will have two stripes of value p_B at $d = \pm w/2$ extending for $w_c/2$ toward the center. To ease the optimization process, instead of having a neat boundary mark, this is linearly smoothed toward the center. Figure 4.10 shows the final effect of painting the boundaries on the potential map. When displaying potential

maps, black correspond to the lowest potential area while yellow is the highest value.



(a) (b)

Figure 4.10: Border painting, displayed in Cartesian 4.10a and road coordinates 4.10b.

Embedding obstacles requires little more work. Assuming that each obstacle is expressed as the collection of contour vertices, their road coordinates are computed with equations (4.12)-(4.13), obtaining $o_r = \{p_1, \ldots, p_n | p_i = \begin{pmatrix} s_i \\ d_i \end{pmatrix}\}$. To handle sparse contours, the obstacles are approximated with their *road-aligned bounding box*, which is a rectangle in road coordinate defined by the two corners:

$$\begin{aligned} p_{\text{bl}} &= \begin{pmatrix} s_{bl} = \min_i s_i \\ d_{bl} = \min_i d_i \end{pmatrix} \\ p_{\text{tr}} &= \begin{pmatrix} s_{tr} = \max_i s_i \\ d_{tr} = \max_i d_i \end{pmatrix} \end{aligned} \tag{4.14}$$

See 4.11 for an example. When embedding the obstacles in the potential, two expedients are used to help the optimization.

First, when the obstacles are located near one side of the reference it's easy to locate the best evasion direction. To help the path converge toward the best avoidance maneuver, the potential is painted asymmetrically, assigning higher cost to the "wrong" evasion side. If the potential assigned to obstacles is p_O,

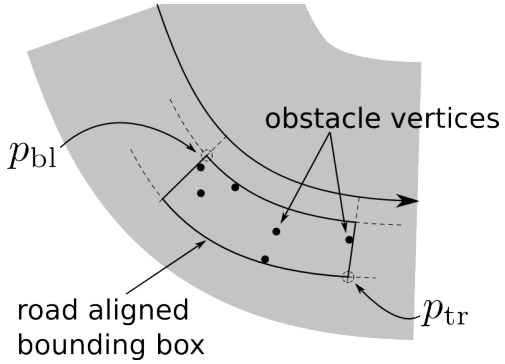


Figure 4.11: Road aligned bounding box.

the obstacle is painted with values that change linearly from ηp_O ($\eta > 1$) to p_O going from one side to the other. Image 4.12 shows an example of asymmetric obstacles painting.

Secondly, to address the issue of limited planning horizon, the potential of the obstacle is back-smoothed up to a distance proportional to the speed of the car. If the obstacles were painted as boxes with sharp boundaries, when planning a path the cost function would assign high cost only to paths long enough to reach the obstacles' zone. This would mean that, even with the obstacles already in the potential map, the car would not react until it's close enough that the planned paths interact directly with the obstacles, possibly resulting in abrupt avoidance maneuvers. Avoiding "last-moment" evasions, requires the path to be aware of the presence of obstacle ahead; this is obtained smoothing the obstacles' potential in the decreasing s direction up to a distance obtained multiplying the current car speed with a configured reaction time t_r. Figure 4.13 shows the effect of increasing speed on the potential smoothing. To further discourage paths in collision course even with far obstacles, a small potential ϵ_O is assigned to $d = [d_{bl}, d_{tr}]$ for $s \leq s_{bl}$.

The exposed procedures permit the definition of the full potential map $\Phi_{i,j}$. The samples are then linearly interpolated to provide access to any (s, d) coordinates. Given the path to evaluate $P = \{\mathrm{z}_1, \ldots, \mathrm{z}_n | \mathrm{z}_i = (x_i, y_i, \theta_i)\}$ final

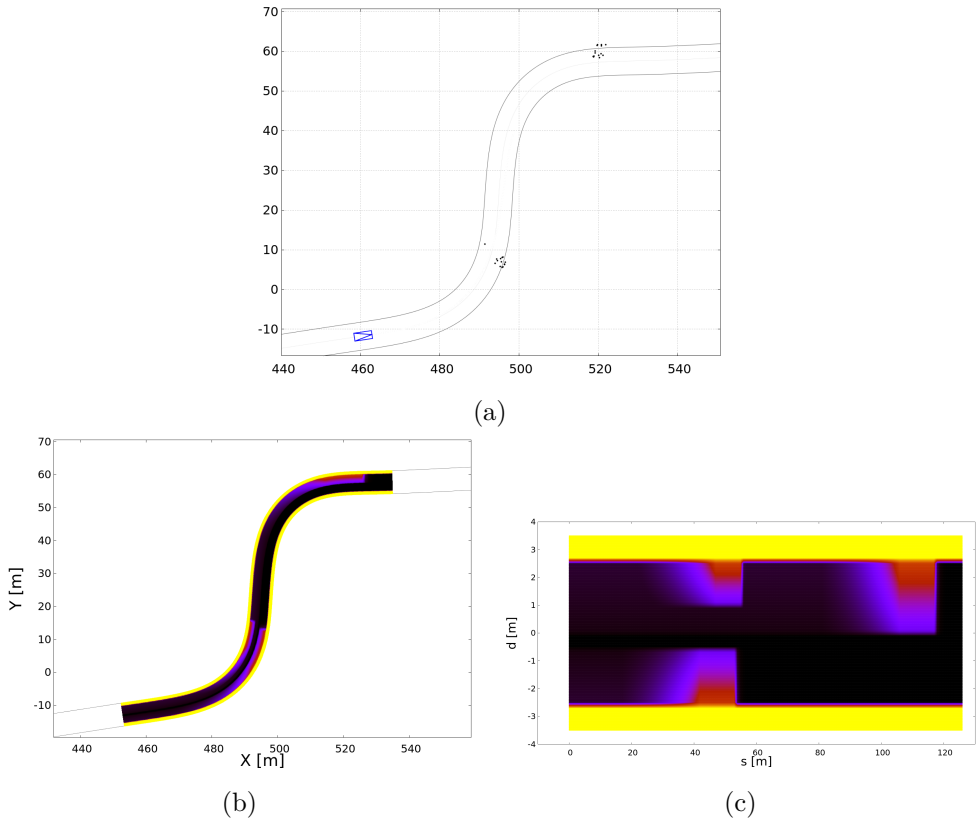
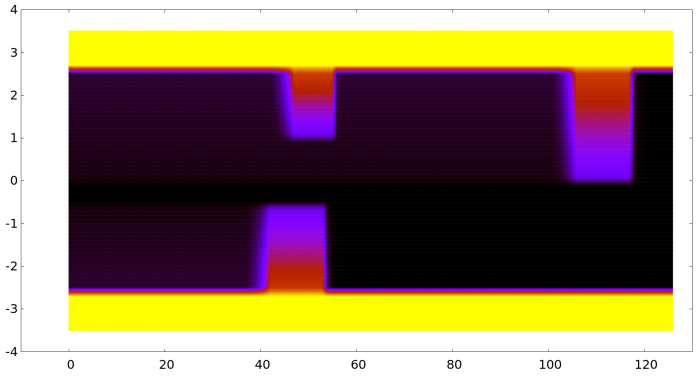


Figure 4.12: Obstacles on the track 4.12aand relative potential in Cartesian 4.12b and road coordinates 4.12c.

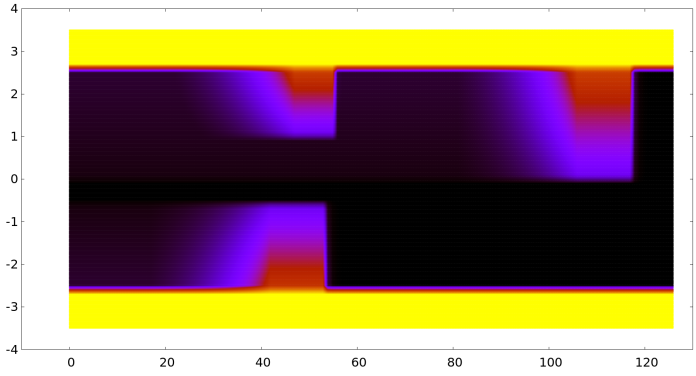
contribution of the potential on J is:

$$\frac{1}{n}\sum_{i=1}^{n}\max\left\{\Phi(x_i,y_i),\Phi(x_i+l_c\cos\theta_i,y_i+l_c\sin\theta_i)\right\} \tag{4.15}$$

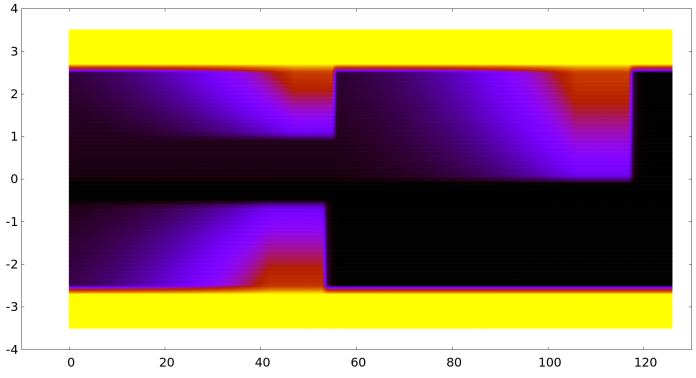
where the function $\Phi(\cdot,\cdot)$ subsume the conversion between Cartesian and road coordinates. The accumulated potential is normalized on the number of states to provide a fair comparison between paths with different number of samples.



(a) Car speed = $0m/s$.



(b) Car speed = $5m/s$.



(c) Car speed = $10m/s$.

Figure 4.13: Effect of car's speed on obstacle painting.

Orientation error

While the potential penalizes trespassing the reference boundaries, more informations can be incorporated to guide the convergence even when the paths are not crossing those boundaries. The problem with using only the potential is that the cost is assigned on a semi-binary basis: positions inside the lane are not penalized, regardless of the relative orientation[1], while faulting the limit results is high cost. This behavior probably reflects in the cost function as a relatively flat region, which is known to make the optimization harder. To compensate for this, an error contribution on the relative orientation between the car and the navigation reference can be introduced.

The dynamic of θ, as prescribed by the equation (4.5), is a straight integration of the control function $\kappa(s)$, so assigning an error on θ can provide a strong signal to accelerate the convergence.

It must be noted that this contribution might be contrasting with the minimization of the accumulated potential: avoidance maneuvers take the car's orientation away from the one of the road. Since the potential encodes informations directly related to *safety*, the orientation errors must be weighted with a proper coefficient $\alpha_o \geq 0$ in order to allow avoidance maneuvers when needed.

The introduced orientation error effectively helps the convergence, but it also introduces a side effect: the car follows the exact same shape of the road. While this can sound pretty natural, when tested on the real car it results in a quite unnatural driving behavior. This feeling comes from the tendency of human drivers to "cut the corners" so, to result comfortable, the car should drive in a similar way.

To achieve this *natural* driving style, the orientation error can be modified introducing a look-ahead distance d_l. For each car pose, the error contribution is computed comparing the orientation the car would have keeping the current curvature constant for d_l meters, with the one of the reference d_l meters ahead

[1] Assuming that also the tip of the car is inside the reference.

from the current position.

Assuming $R(s)$ represents the reference point with curvilinear abscissa s, and $\bar{s}_i$ being the curvilinear abscissa corresponding to the i^{th} path sample z_i, the error contribution with look-ahead is:

$$\frac{\alpha_o}{n}\sum_{i=1}^{n}\left|\Lambda\left(\theta_i + d_l\kappa(i\Delta s), R(\bar{s}_i + d_l)\right)\right| \tag{4.16}$$

Where $\Lambda(a,b)$ computes the distance between a and b taking into account the $\mathbb{S}^1$ topology [97].

Comfort

The requirement of the path being *comfortable* is the most troublesome to express, as there's not exact definition of what "comfort" is. There is, however, consensus in acknowledging that lateral acceleration and jerk are related to the perceived comfort [99]. Acceleration and jerk (the acceleration's derivative) depend, through the car speed, on the path curvature and curvature derivative, therefore it's natural to impose some regularization on the function $\kappa(s)$. This regularization shall not prevent the car from correctly following the road and avoid obstacles, hence it's weight should be adjusted accordingly. The comfort contribution is:

$$\frac{\alpha_c}{s_f}\int_0^{s_f}|\kappa(s)| + \alpha_{c,\kappa'}|\kappa'(s)|\,ds \tag{4.17}$$

where again there's the normalization coefficient $1/s_f$ to properly compare path of different lengths.

Progress along the road

The great flexibility of variational methods reside in the possibility of incorporating the goal in the cost function. This makes possible to express preferences for certain conditions, as opposed as fixing mandatory final poses or regions. When not fixed final position/region is naturally available, the intermediate goal it's just to progress along the road and this can be easily embedded in

J. To encourage the advancement, J assigns a reward (negative cost) at the difference between the curvilinear abscissa of the last and first path samples, as measured by their projection on the reference track. Note that, unlike rewarding the total path length, this goal doesn't encourage long paths that oscillate within the reference.

Assuming $\bar{s}_i$ being the curvilinear of the i^{th} path sample, out of n, the reward is:

$$\alpha_r(\bar{s}_n - \bar{s}_1)$$

With $\alpha_r \leq 0$ to properly balance advancement against the other requirements.

This simple rewarding, however, hides potential problems. Pathological road shapes, like the one shown in figure 4.14, might induce the path to "jump" over the reference boundaries to gain a substantial reward (the potential p_B must be finite for optimization-stability reasons). This can be prevented with the indicator function:

$$\mathbb{I}_{(\bar{s}_i, \bar{s}_{i+1})} = \begin{cases} 1 & \text{if } \bar{s}_{i+1} - \bar{s}_i > \text{Th} \\ 0 & \text{otherwise} \end{cases}$$

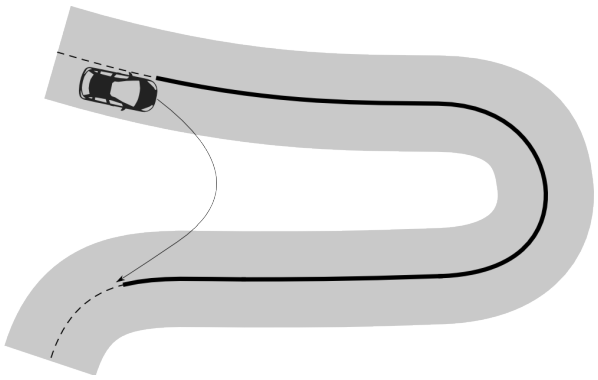


Figure 4.14: Example of "jump": cutting along the arrow, the car would gain the bonus corresponding to the solid line.

Including the "anti-jump" contribution, the final reward is:

$$\alpha_r(\bar{s}_n - \bar{s}_1) + M \sum_{i=1}^{n-1} \mathbb{I}_{(\bar{s}_i, \bar{s}_{i+1})} \tag{4.18}$$

Where $M > 0$ is the penalty assigned to paths that jumps across the boundaries. Aiming to prevent only pathological cases, M can be chosen to be as large as it's required to counter the reward gain from a jump.

Optimization

The previous sections described the components required for a cost function J that comply with the requirements 4.3.2. Aggregating all the contribution, the final form for J is:

$$\begin{aligned} J(p; R, O_r) = \bar{J}(P, \kappa; R, O_r) = \frac{1}{n} \sum_{i=1}^{n} \max \{\Phi(x_i, y_i), \Phi(x_i + l_c \cos\theta_i, y_i + l_c \sin\theta_i)\} \\ + \frac{\alpha_o}{n} \sum_{i=1}^{n} |\Lambda(\theta_i + d_l \kappa(i\Delta s), R(\bar{s}_i + d_l))| \\ + \frac{\alpha_c}{s_f} \int_0^{s_f} |\kappa(s)| + \alpha_{c,\kappa'} |\kappa'(s)| \, ds \\ + \alpha_r(\bar{s}_n - \bar{s}_1) + M \sum_{i=1}^{n-1} \mathbb{I}_{(\bar{s}_i, \bar{s}_{i+1})} \end{aligned} \tag{4.19}$$

Where it's implicit that P and κ are functions of the parameter vector p, as described in 4.3.1.

To enforce the feasibility of the path, it's simply necessary to introduce bounds constraints in the optimization process. Assuming the parameters to be $p = (\kappa_0, \ldots, \kappa_{N-1}, s_f)^\top \in \mathbb{R}^{N+1}$, a lower and upper bounds are defined as:

$$p_{lb} = (\kappa_{\min}, \ldots, \kappa_{\min}, l_{\min})^\top \in \mathbb{R}^{N+1} \tag{4.20}$$

$$p_{ub} = (\kappa_{\max}, \ldots, \kappa_{\max}, l_{\max})^\top \in \mathbb{R}^{N+1} \tag{4.21}$$

And the feasibility constraints become:

$$p_{lb} \leq p \leq p_{ub} \tag{4.22}$$

where the inequality is intended element-wise. Many nonlinear optimization algorithm can easily handle bound constraints [100].

Since κ has only a piece-wise definition and the ODE (4.5) doesn't admit closed form solution, there is no readily available gradient expression for $J(p)$. Therefore, to perform optimization, two choices are available: computing numeric gradient or use algorithms that don't require the gradient.

To compute the numeric gradient of the function, many formulas are available with different degrees of accuracy. High order formulas provide greater accuracy, given that the function's higher derivatives are sufficiently smooth [101]. Since there are no guarantees that J satisfy such conditions, the preferred formula is the cheap *forward difference*, that requires only $N+2$ function evaluations. Indicating with p_i, $i = 1, \dots, N+1$ the components of p and with $\bar{p}$ the parameter being evaluate, the approximation formula is:

$$\nabla J|_{p=\bar{p}} = \begin{pmatrix} \frac{\partial J}{\partial p_1}\Big|_{p=\bar{p}} \\ \vdots \\ \frac{\partial J}{\partial p_{N+1}}\Big|_{p=\bar{p}} \end{pmatrix}$$

$$\frac{\partial J}{\partial p_i}\Big|_{p=\bar{p}} \approx \frac{J(\bar{p} + h_i \boldsymbol{e}_i) - J(\bar{p})}{h_i} \tag{4.23}$$

where $\boldsymbol{e}_i$ is the i^{th} vector of the natural basis of $\mathbb{R}^{N+1}$, and h_i a sufficiently small positive scalar.

The function J is generally not convex, so the optimization cannot be performed with a local method, but a global optimization algorithm is needed. The optimization results will be reported for two different algorithms: CRS and StoGO.

Controlled Random Searches (CRS), is a global derivative-free optimization algorithm [102, 103]. It's sometimes compared to evolutionary algorithm, in that it starts with a random set of samples and evolve them with local transformation in a way similar to the Nelder-Mead algorithm [104].

Stochastic Global Optimization (StoGO) is another global algorithm that makes use of gradient information [105, 106]. It works by iteratively dividing

# J eval	CRS		StoGO	
	8 parameters	16 parameters	8 parameters	16 parameters
8000	-0.051116	-0.023040	-0.026461	0.017616
4000	-0.033066	-0.005641	-0.018617	0.017616
2000	-0.019973	0.050888	-0.016061	0.017616
1000	0.024884	0.102560	-0.016061	0.075202
500	0.080545	0.190447	-0.016061	0.510867

Table 4.1: Scenario-I optimization results.

# J eval	CRS		StoGO	
	8 parameters	16 parameters	8 parameters	16 parameters
8000	-0.029977	-0.029928	-0.097759	-0.090566
4000	-0.029942	-0.029499	-0.096771	-0.083746
2000	-0.029499	-0.029499	-0.091908	-0.083746
1000	-0.029499	-0.029499	-0.083993	-0.083746
500	-0.029499	-0.029499	-0.083993	-0.063290

Table 4.2: Scenario-II optimization results.

the bound-constrained search space into smaller hyper-rectangles via branch-and-bound. Each subspace is searched with a variant of the BFGS local search [107].

The two optimization algorithm are compared on 3 test cases, selected to provide different situations:

Scenario-I : a long double turn, picture 4.15a.

Scenario-II : a straight road segment, picture 4.15b.

Scenario-III : a straight road segment with an obstacle in front of the car position, picture 4.15c.

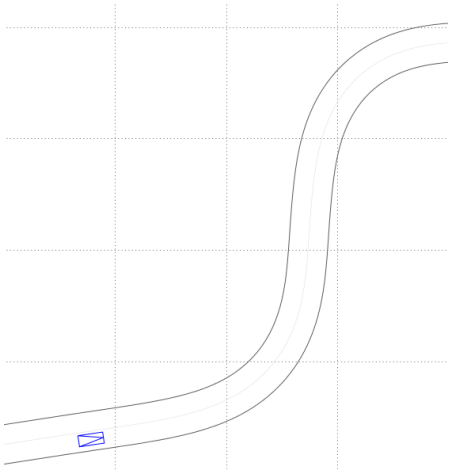
# J eval	CRS		StoGO	
	8 parameters	16 parameters	8 parameters	16 parameters
8000	0.040723	0.058243	0.075119	0.081119
4000	0.054579	0.093885	0.079349	0.087597
2000	0.078608	0.192431	0.079936	0.090856
1000	0.162671	0.280404	0.080588	0.091706
500	0.277530	0.360404	0.080588	0.091706

Table 4.3: Scenario-III optimization results.

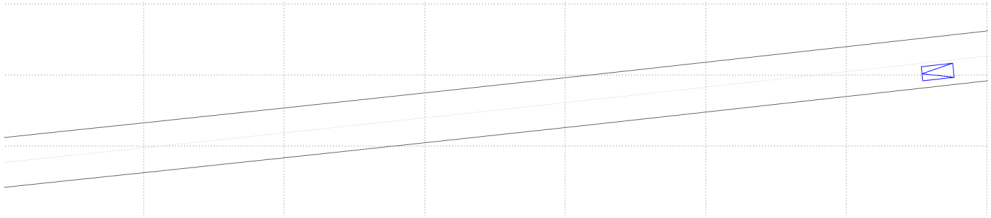
The comparison is based on the average final value obtained for J varying the number of allowed cost function evaluations (considering also the evaluation required to numerically compute ∇J) and the size of the optimized vector (which is 1 plus the number of curvature control points). Note that the actual cost value it's not meaningful because it depends on many configuration parameters (tuned to achieve the intended path shape) and the value of the true optimum in unknown.

Tables 4.1–4.3 shows the results obtained on the three test cases. The first consideration is that allowing too much control over the curvature function, increasing the size of the optimized vector, makes the problem harder without providing apparent benefits. Another consideration is that StoGO, using the derivative information, is much less sensitive to both the maximum number of function evaluations and the size of the optimized vector. With the exception of scenario-II, where StoGO performs much better than CRS, the general tendency is that CRS outperforms StoGO with enough computation power. With a `C++` implementation on a `Intel(R) Core(TM) i7-4790` CPU, to run at 10Hz the real-time computation limit lies around 2000 evaluations, which is roughly where the performance of the two algorithm match. For these reasons, considering the improved robustness to compute power limitations, the solution algorithm of choice is StoGO. Moreover, for the lack of benefits, the path is parametrized with 7 curvature control points, for a total size of 8.

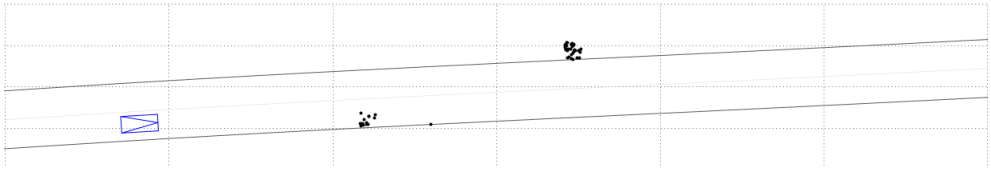
Qualitative results will be presented for the path planning problem, solved with the selected parametrization and algorithm.



(a) Scenario-I



(b) Scenario-II



(c) Scenario-III

Figure 4.15: Test scenarios. The blue rectangle represents the car with the tip of the triangle pointing in the heading direction. The black line indicates the road borders and the black circles are simulated obstacle's vertexes.

4.3.3 Improving stability and planning horizon: path chunks

The introduced procedure can successfully plan local path that progress along the navigation reference in a collision-free and comfortable way. There are, however, some issues that must be addressed.

First, since the optimization is bounded by the computation time available and the function evaluation cost depends on the length of the path, there must be a reasonable bound on the maximum length of the planned path. This length is also referred to as the *planning horizon*, meaning that it's the (distance) limit beyond which the planner is *unaware* of the environment. While the obstacle embedding has been specifically designed to tackle this problem, in some cases having a long planned path is beneficial. This is the case of high-speed driving, where a longer preview is needed to properly stabilize the vehicle on the path.

Secondly, a proper strategy for *replanning* must be defined. The planning environment is highly dynamic, both because of its very nature and the use of on-board sensors, thus the car must properly react to changes and this requires an adaptation of the plan: a replan. The basic strategy for replanning is to compute a new path from scratch every planning iteration. While it would provide the frame-by-frame optimal path, when tested on the car this results in a very poor strategy. Each replanning is likely to produce, possibly minimal, modifications to the path. Even when the environment is static, due to the different car positions, the "old" path will not generally be in the span of possible new paths since the chosen parametrization somehow restricts the set of feasible shapes. Due to the low-level control action, each path modification results in a slight adjustment of the steering that is , in turn, perceived as unnecessary and detrimental for the comfort feeling.

To provide solution to the highlighted issues, the final path is obtained piecewise with the subsequent planning of two shorter path chunks. Once the first path P_1 has been planned, the following invocations of the planning procedure will perform different actions. First, P_1 is checked to see if it's still consistent with the potentially new environment, as indicated by a large

change in the cost assigned to the path. If P_1 it's still acceptable, a new path P_2 will be computed starting from the last point of P_1. Subsequently, if P_1 is valid and a path P_2 already exists, the global optimization process is resumed to further improve P_2. When the car reaches P_2, the first chunk is discarded and the second will assume the role of P_1. If instead, at any moment, P_1 is found to be invalid all the paths are discarded and a new one is planned from scratch. This procedure is detailed in algorithm 4.2.

Algorithm 4.2 Chunk planning algorithm

Require:

PLAN ▷ local path planning procedure

1: **function** CHUNKPLANNING
2: **if** $\exists P_1 \wedge$ its cost has changed **then**
3: $P_1, P_2 \leftarrow \emptyset$
4: **end if**
5: **if** $\exists P_1 \wedge \exists P_2$ **then**
6: improve P_2
7: **else if** $\exists P_1$ **then**
8: $P_2 \leftarrow$PLAN(from the end of P_1)
9: **else**
10: $P_1 \leftarrow$PLAN(from the current position)
11: **end if**
12: **return** $P_1 \bigcup P_2$
13: **end function**

The benefits of such planning strategy are manifold.

First the planned path is maintained fixed as long as it's still consistent with the environment. The consistency is checked in terms of assigned cost. Filtering the small variations of cost, only large change induced by modifications of obstacles or road borders trigger a full replanning. This replanning strategy tries to keep the path fixed, resulting in a smooth driving and reacting only when changes in the environment pose potential threat to safety.

Moreover, the concatenation of multiple paths results in a longer planning horizon, providing adequate preview also for high speed driving. It must be noted that planning a long path as two short pieces, it's not equivalent to perform it in a single pass. In particular, the solution might be sub-optimal because the second path is obtained from the fixed final point of the first one, which was optimal only respect the previous portion of potential. While lacking theoretical optimal-guarantees this technique it's still very useful, especially when combined with the used replanning strategy. When keeping the path fixed, having a further separate path head avoids the car coming dangerously close to the end of the planned reference before computing a new one.

Lastly, another important advantage of this strategy is that, with the except of the very first one, each path is effectively computed with a time bound larger than the one assigned to a single execution. In a real-time context, this ameliorate the potential problems deriving from the limited availability of computational time.

4.3.4 Results

The introduced path planning method is consistently able to find feasible paths that smoothly adapt to road navigation and obstacle avoidance.

Figure 4.16 show the planning results for the test scenarios selected for the optimization comparison.

In picture 4.17 is shown the effect of the orientation error contribution on the final solution. Note how, using only the potential function and the comfort contribution, the path proceeds straight until it "bounces" on the borders. Properly blending the orientation contribution helps the path to negotiate the turn and, using the look-ahead distance, it finally achieve a natural "corner-cutting" behavior.

The approach can successfully adapt to a wide variety of planning situations. Figure 4.18 shows examples of navigation with obstacle avoidance in a wide track. Image 4.19 depicts navigation results in case of narrow reference tracks.

A big advantage of the introduced approach is that it can provide a comfortable driving even when the navigation reference is noisy. Explicitly including the comfort in the cost function allows the algorithm to find a nice balance between matching the reference shape and avoiding continuous oscillations. Image 4.20 shows a comparison between the planned curvature and the one of the intentionally noisy reference, it can be seen how the path filters the unnecessary oscillations.

Using a properly shaped potential function, the proposed method can successfully plan also lane change maneuvers, depicted in figure 4.21. In case of lane change, the potential is painted with a high value for the current lane, gradually decreasing toward to the target lane, shown in picture 4.22.

This section introduced a new approach to path planning in urban environment based on variational methods. The algorithm is able to plan local paths that are feasible, collision-free and follow a navigation reference. With the introduction of a carefully designed cost function, this approach removes the need of specifying intermediate goal positions leading to a natural driving style. Explicitly including the comfort in the optimized function, this planner is able to effectively smooth the potential noise that affect the navigation reference when it's estimated from sensory data. The proposed approach has been implemented in `C++` and deployed on the DEEVA platform, enabling a consistent real-time path planning with computation time as low as 30–40ms on a `Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz`.

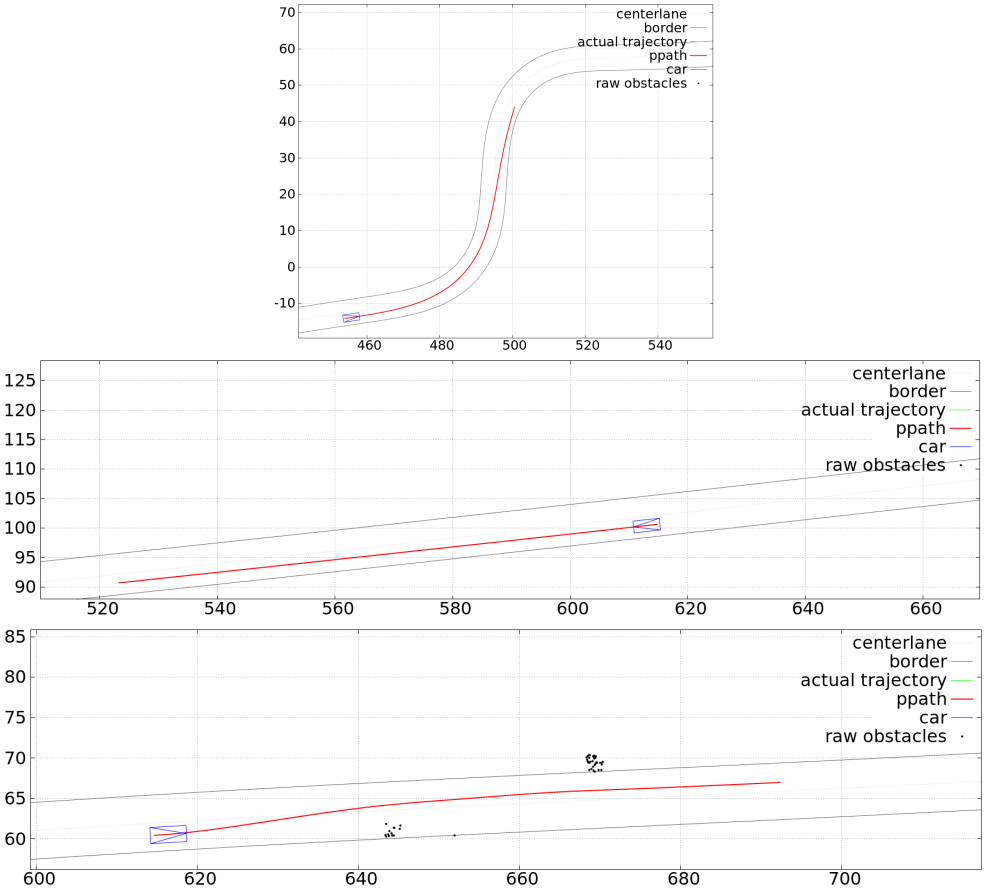
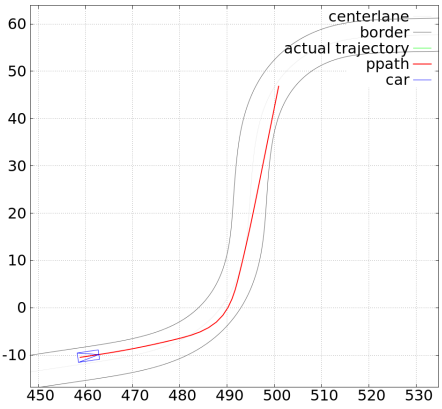
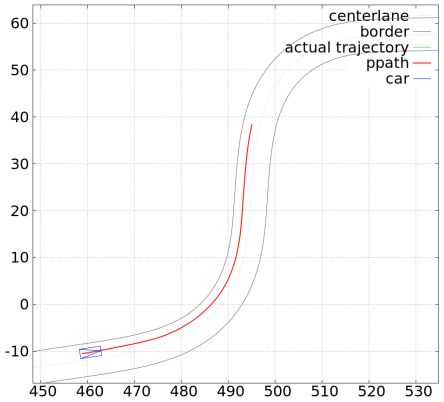


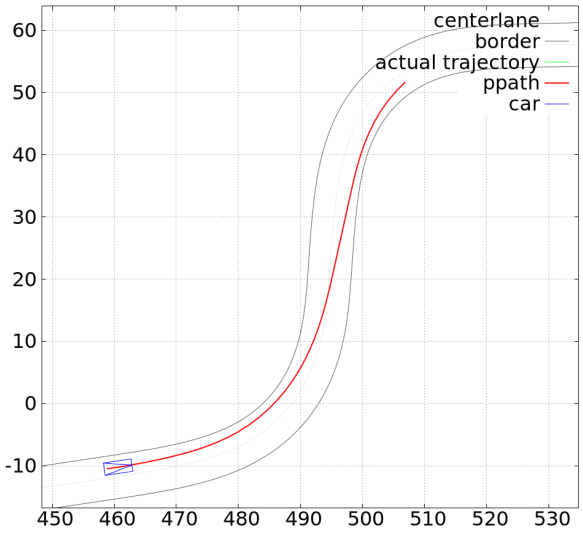
Figure 4.16: Qualitative results on the test scenarios used in the previous section.



(a) No orientation error.



(b) Simple orientation error.



(c) Orientation error with look-ahead distance.

Figure 4.17: Effect of the orientation error on the planned path.

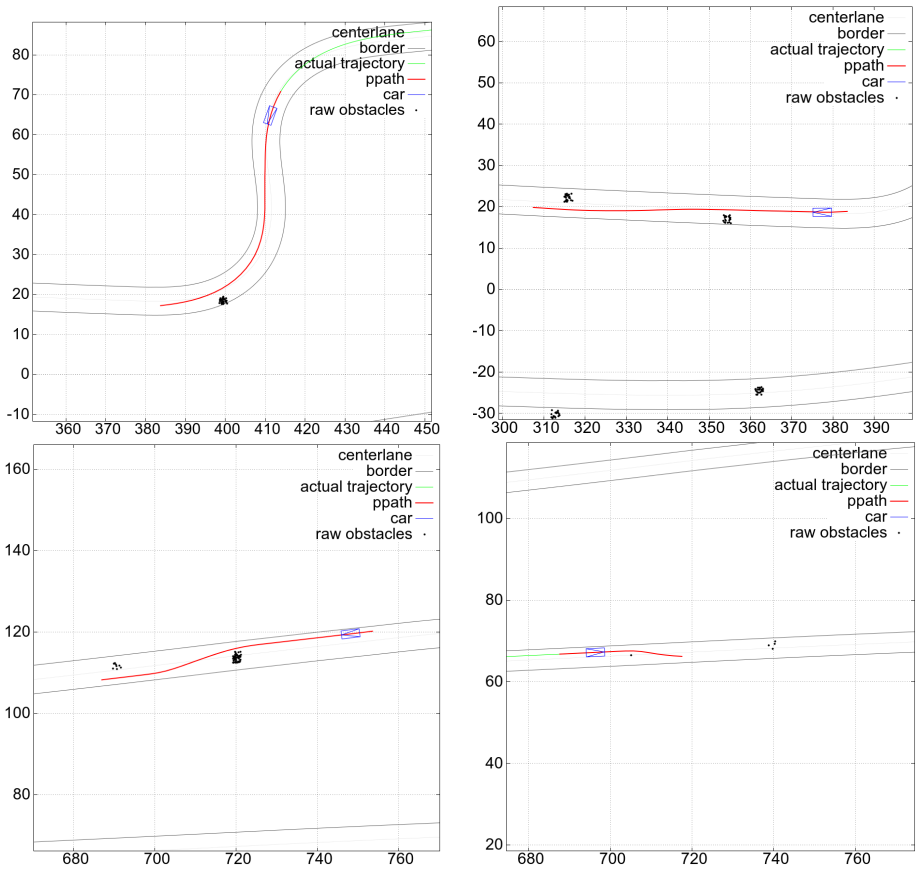


Figure 4.18: Obstacle avoidance results navigating a wide reference track.

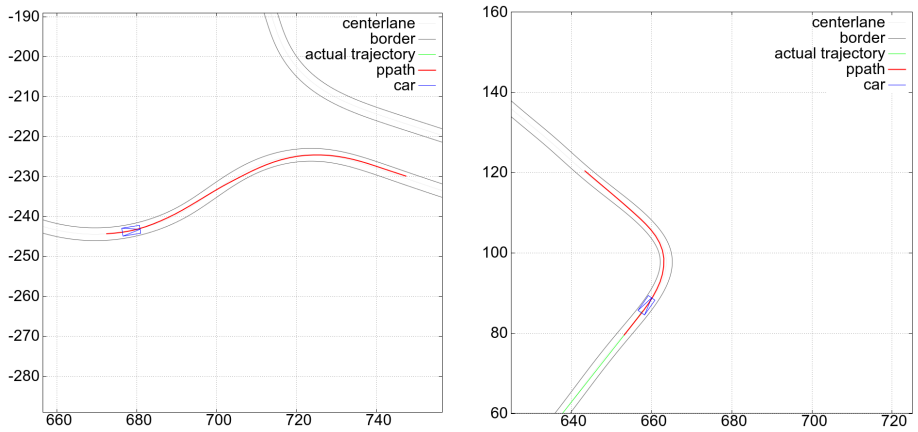


Figure 4.19: Navigation results with a narrow reference track.

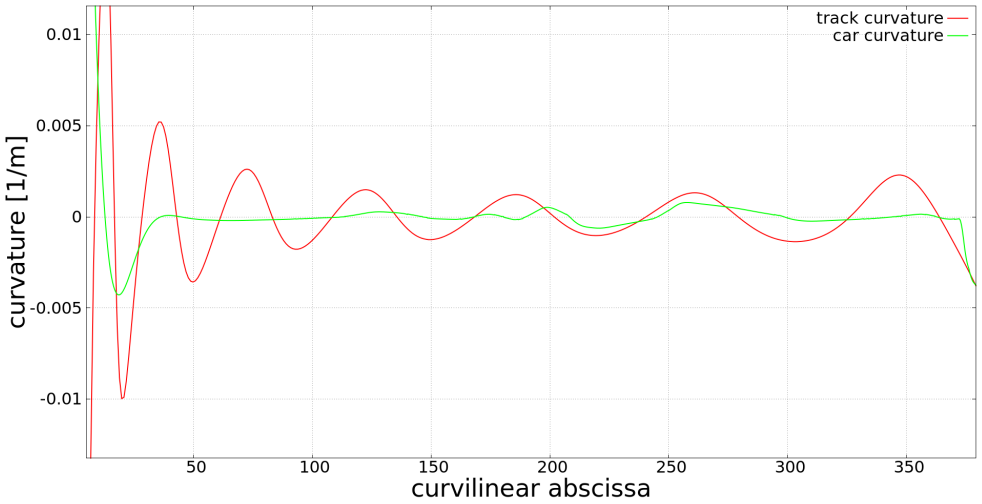


Figure 4.20: Smoothing effect on a noisy reference.

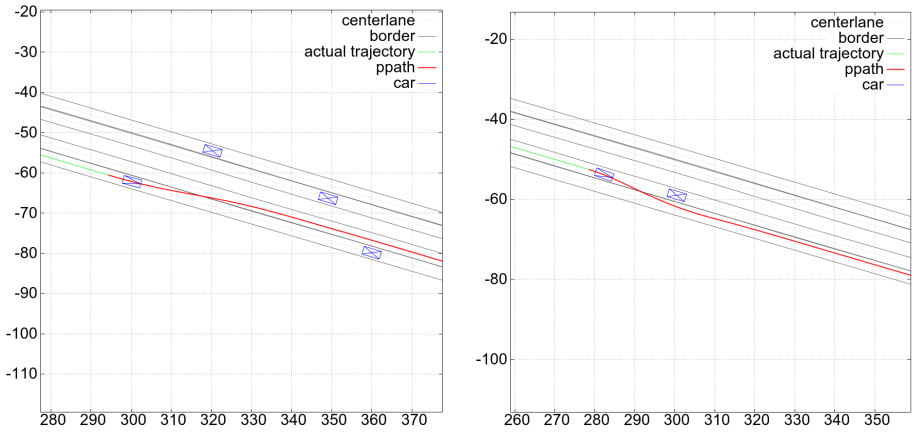


Figure 4.21: Lane change maneuvers.

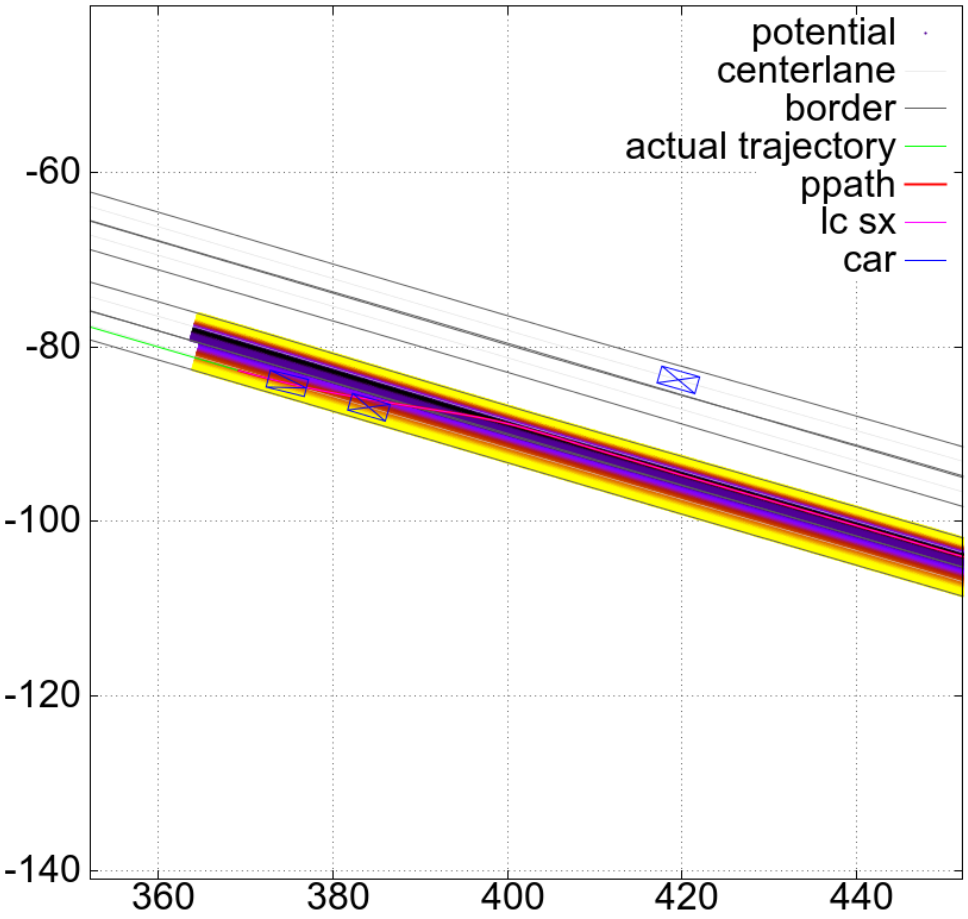


Figure 4.22: Potential map used to perform lane change maneuvers.

4.4 Speed planning

Speed planning is the procedure that complements the path planning in computing a trajectory with the speed tuning approach. Once a path in the configuration space $\pi : [0,1] \to SE(2)$ is fixed, speed planning seeks a *timing function* $\sigma : [0,T] \to [0,1]$, that specifies the speed the path should be traversed with. With reference to the car kinematics (4.1), this is equivalent to finding an acceleration function $a(t)$.

The speed planning takes in consideration the dynamics neglected in (4.5):

$$\dot{v} = a(t) \quad ,\forall t \in [0;T] \tag{4.24}$$

$$a_{\min} \leq a(t) \leq a_{\max} \quad ,\forall t \in [0;T] \tag{4.25}$$

where v is the longitudinal speed of the car and $a(t)$ the applied acceleration at time t and T it's a chosen time planning horizon. Since the shape of the path is already fixed, the x, y and θ coordinates are not relevant anymore and the car is assumed to be an entity moving along an established curve, with relevant states being now the speed v and the curvilinear abscissa s. Therefore, the new configuration space is $(v,s)^\top \in \mathbb{R}^2$ and the differential constraints are:

$$\begin{cases} \dot{v} = a(t) \\ \dot{s} = v \end{cases} \tag{4.26}$$

The classic motion equations provide the explicit solution to the state evolution:

$$v(t) = v_0 + \int_0^t a(\tau) d\tau \tag{4.27}$$

$$s(t) = s_0 + \int_0^t v(\tau) d\tau \tag{4.28}$$

where v_0 and s_0 are the initial values for the state variables.

Beyond the limited longitudinal acceleration expressed in (4.25), there is another constraint due to an effect not modeled in the kinematic model, but

that can't be neglected due to safety considerations: limited lateral acceleration. This constraint is related to vehicle handling: the tires can exert only a limited amount of lateral force due to their cornering stiffness [108, 109]. In terms of circular motion, this translates in a maximum value of lateral acceleration that can be achieved by the car. This acceleration, in turns, depends on the speed and the curvature applied to the vehicle:

$$a_y = v^2|\kappa| \leq a_{y,\max} \tag{4.29}$$

Considering the curvature to be fixed, the speed of the vehicle must satisfy:

$$v \leq \sqrt{\frac{a_{y,\max}}{|\kappa|}} \tag{4.30}$$

assuming both $a_{y,\max} \geq 0$ and $v \geq 0$.

Just like the curvature determines the path, the acceleration function $a(t)$ determines the evolution of the speed and curvilinear abscissa, thus the speed planning reduces to finding a proper acceleration reference value. This acceleration should be chosen such that:

1. The car abide the speed limit laws.

2. The car doesn't collide with other cars and abide the safety distance.

3. It respect the limited acceleration constraint (4.25).

4. The speed of the car satisfy (4.30).

5. The resulting driving is comfortable.

This problem is known to be easier to solve than path planning, thus the proposed solution approach is based on set of *reactive* speed policies. Details are given in the next sections.

4.4.1 Speed limits

The requirements of abiding the law speed limits and not exceeding the maximum lateral acceleration both result in a *upper bound* for the speed of the car. This speed limit is associated with a particular position on the road or on the planned path, so the resulting constraint is:

$$0 \leq v(t) \leq v_{\max}(s(t)) \tag{4.31}$$

for an appropriate function $v_{\max}(s)$.

The function $v_{\max}(s)$ aggregates three different contributions:

- The speed limit associated with road segment the car is navigating v_{law}.
- The limit from the planned path curvature κ_p.
- The limit from the road curvature κ_r.

While the road curvature will not be, in general, the exact curvature the car will follow, it still gives a very good idea of the speed a turn should be approached with. Assuming that the curvilinear abscissa is consistent for both the road and the path, the function $v_{\max}(s)$ can be chosen to be:

$$v_{\max}(s) = \min\left\{v_{law},\ \sqrt{\frac{a_{y,\max}}{|\kappa_r(s)|}},\ \sqrt{\frac{a_{y,\max}}{|\kappa_p(s)|}}\right\} \tag{4.32}$$

where the planned path contribution is included only when applicable. Enforcing the minimum upper bound will ensure that they are all satisfied. Image 4.23 shows an example of reference track with relatives speed limits.

Note that (4.32) express a mandatory speed limit that either results from law prescription or vehicle handling limitations, but that doesn't take into account the comfort of driving. A strategy to seamlessly integrate the comfort requirements is to define a secondary speed limit for the purpose. Since humans are not sensitive to absolute speed, the comfort speed limit can be derived from a lower lateral acceleration limit $a_{y,\text{comf}}$:

$$v_{\text{comf}}(s) = \min\left\{\sqrt{\frac{a_{y,\text{comf}}}{|\kappa_r(s)|}},\ \sqrt{\frac{a_{y,\text{comf}}}{|\kappa_p(s)|}}\right\} \tag{4.33}$$

This speed limit is not mandatory, so no violent braking/acceleration actions should be applied to strictly follow its value.

The speed planning suffers from the limited horizon problem, just like the path planning did. For example, if the speed limit contains a sudden drop (maybe due to a tight turn), the car will react only when the state s will reach that point within the time horizon T. This situation might result in a unnecessarily late and violent braking action, or even worse if T is not long enough. To prevent this dangerous situations, the speed limits are preprocessed with a "backward smoothing" through a proper acceleration value a_s. Considering a sampled version of the generic speed limit $v_x(s)$:

$$v_{x,i} = v_x(i\Delta s) \quad , i = 1, \dots, N \tag{4.34}$$

the smoothing replaces each sample with the minimum between its value and the speed needed to arrive at the next sample's speed with acceleration a_s. The equation is:

$$v_{x,i} = \min\left\{v_{x,i},\ \sqrt{v_{x,i+1}^2 - 2a_s\Delta s}\right\} \quad , i = N-1, N-2, \dots, 1 \tag{4.35}$$

This smoothing ensures that the progression of speed limits can be followed applying a brake value of at most a_s.

The profiles $v_{\max}$ and v_{comf}, due to their distinct nature, should be smoothed with different braking values. The maximum speed $v_{\max}$ can be smoothed with a value close to $a_{\min}$ because it represents the feasibility limit. On the other hand, v_{comf} should be smoothed with the braking value a_{comf} that is considered to be *comfortable* for the passengers. Image 4.24 provides an example of speed limit smoothed with this technique.

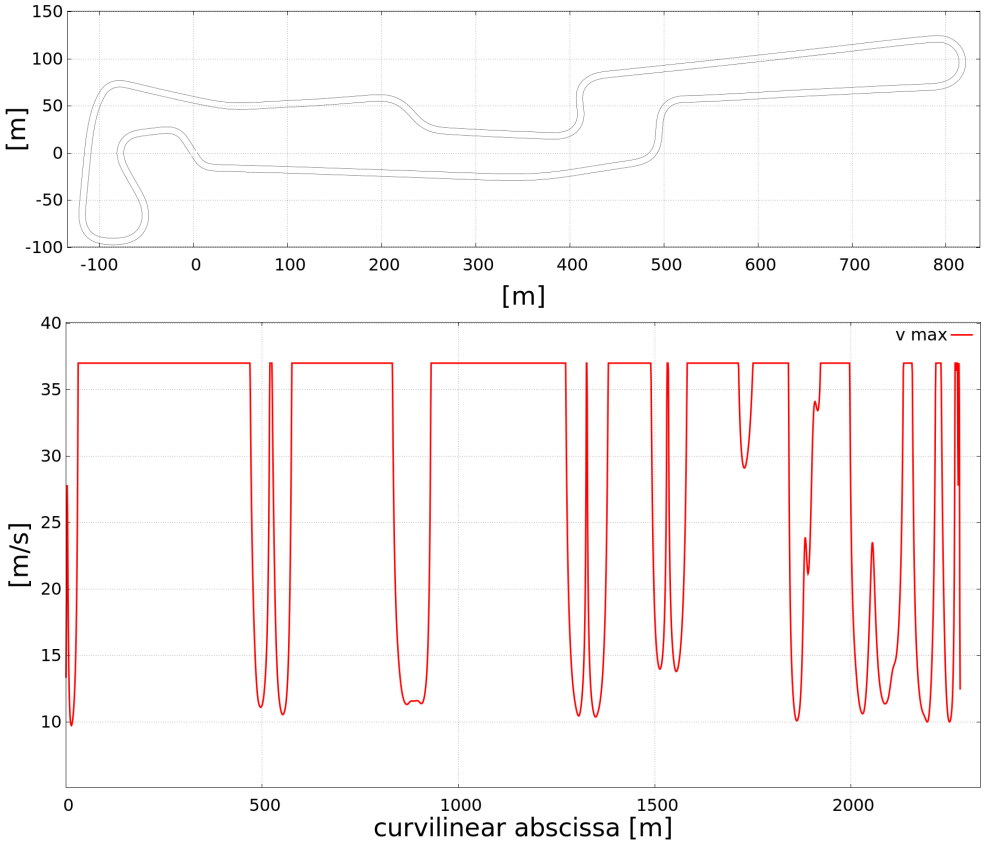


Figure 4.23: Reference track with speed limit. The curvilinear abscissa is in the origin of the Cartesian coordinates, increasing counter-clockwise. The speed limit is set to $37m/s$ and lowered to enforce a maximum lateral acceleration of $6m/s^2$. The shape is jagged because the track is obtained from a recorded drive so the curvature contains some oscillations.

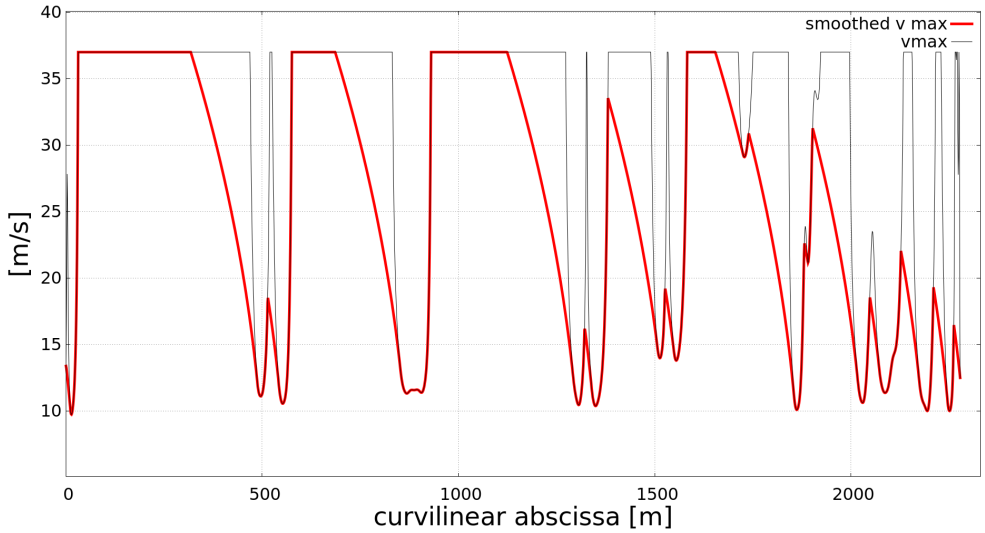


Figure 4.24: Speed limit with backward smoothing.

4.4.2 Speed policies

The previous section showed how most requirements can be expressed as static speed limits, changing with the curvilinear abscissa but fixed in time. The other situation that needs to be considered is the one deriving from the presence of *road obstacles*. Other traffic participant are entities that change state with time and, thus, the generated constraints must be time-dependent too.

The two type of constraints, static and time-varying, are quite different and this reflects also a distinction in the way they should be handled. In particular, human drivers react differently to an upcoming turn and the presence of a slow vehicle in the road. To allow for a clean separate tuning of the two different behaviors, the proposed solution approach is based on a set of reactive policies. The rationale behind it is that each policy is a simple method, devised to deal with a specific defined case, that can be configured independently from the others. Properly aggregating policies for the cases of interest, the emerging behavior adjust smoothly to the conditions.

Three distinct policies have been identified:

Speed limit keeping : it selects the acceleration according to the static speed limit.

Adaptive cruise control : it selects an acceleration that adjust the ego-speed to that of the leading vehicles and keeps a safe distance.

Complete stop : it takes the car to stop at a certain distance. This additional case is needed to complement the speed limit keeping policy.

These policies are intentionally very simple, which translates to *easy to tune*, and they compute an instantaneous reference value for the car's acceleration, assumed to be kept constant until the next planning iteration. The computation doesn't explicitly consider the acceleration's bounds, so the final

value is clipped in the feasibility interval:

$$a = \begin{cases} a_{\min} & , \text{ if } \bar{a} < a_{\min} \\ \bar{a} & , \text{ if } a_{\min} \leq \bar{a} \leq a_{\max} \\ a_{\max} & , \text{ otherwise} \end{cases} \tag{4.36}$$

With a proper tuning of the policies' parameters this clipping is enough to obtain satisfactory results.

The policies will be detailed next, providing also a safe integration scheme.

Speed limit keeping

The speed limit keeping (SLK in the following) policy deals with the static speed limits $v_{\max}$ and v_{comf}. Adopting a control-system like approach, this policy computes an instantaneous reference for the acceleration as a weighted sum of speed errors between the ego speed and the speed limits. To smoothly adapt to the changes in the speed limit, the reference is not computed only for the current position, but using all the values within a look-ahead distance.

Assuming that v_{car} is the ego-speed, $v_{\max,i} = v_{\max}(i\Delta s)$ and $v_{\text{comf},i} = v_{\text{comf}}(i\Delta s)$ are the sampled speed limits, $\bar{i}$ is the sample index that correspond to the current car position and M is the number of samples needed to cover the look-ahead distance, the acceleration is computed according to:

$$\begin{aligned} a_{\text{SLK}} = \frac{1-\gamma_{\text{s}}}{1-\gamma_{\text{s}}^{M+1}} \sum_{i=\bar{i}}^{\bar{i}+M} \gamma_{\text{s}}^{i-\bar{i}} \Big[& K_{\text{s},m}^{+} (v_{\max,i} - v_{car})^{+} + K_{\text{s},m}^{-} (v_{\max,i} - v_{car})^{-} \\ & + K_{\text{s},c}^{+} (v_{\text{comf},i} - v_{car})^{+} + K_{\text{s},c}^{-} (v_{\text{comf},i} - v_{car})^{-} \Big] \end{aligned} \tag{4.37}$$

where $()^{+}$ and $()^{-}$ indicates respectively the positive and negative part of a real number such that $x = (x)^{+} + (x)^{-}$. Different gains for positive and negative speed errors permit to achieve the required asymmetry: the car should firmly brake for exceeding $v_{\max}$, but should not equally accelerate to achieve its value. The parameter γ_{s}, chosen to be $0 < \gamma_{\text{s}} < 1$, is a *discount factor* to weight more the speed limit close to the car and exponentially less the far values. To keep

the resulting acceleration consistent also when changing the look-ahead M, the summation is normalized with $(1-\gamma_{\mathrm{s}})/(1-\gamma_{\mathrm{s}}^{M+1})$ which is the value of the truncated geometric series.

Adaptive cruise control

The adaptive cruise control (ACC in the following) policy is designed to deal with road obstacles. It's assumed that a road obstacle o_i is provided with informations about it's curvilinear abscissa $s_i(t)$ and speed $v_i(t)$ along the road. It's also assumed that a reasonable estimate of its future states is available for $t \in [0, T]$.

The function $d_S(v)$ is used to indicate the safety distance required given the speed v. This function must be selected to provide a reasonable time margin for the vehicle to react to events, an example of such function is:

$$d_S(v) = \max\{d_{min},\, v\} \tag{4.38}$$

which is the distance covered during 1 second at constant speed v, saturated to avoid having a null safety distance. To avoid having the car oscillating around the required safety distance, the additional contribution of the speed error between the ego and leading vehicle is added.

To compute the acceleration reference, a time step Δt is established and all the leading road obstacles are considered within enough steps to cover the time horizon T. Assuming n obstacles and $m = \lceil T/\Delta t \rceil$ preview steps, v_{car} the speed of the car and $s_{car}(t)$ that car's curvilinear abscissa at time t (assuming constant speed), the acceleration is computed according to:

$$\begin{aligned} a_{\mathrm{ACC}} = \min_{\substack{i=1,\dots,n \\ j=0,\dots,m}} \gamma_{\mathrm{a}}^{j} \Big[& K_{\mathrm{a},v}^{+}(v_i(j\Delta t) - v_{car})^{+} + K_{\mathrm{a},v}^{-}(v_i(j\Delta t) - v_{car})^{-} \\ & + K_{\mathrm{a},d}^{+}(s_i(j\Delta t) - d_S(v_i(j\Delta t)) - s_{car}(j\Delta t))^{+} \\ & + K_{\mathrm{a},d}^{-}(s_i(j\Delta t) - d_S(v_i(j\Delta t)) - s_{car}(j\Delta t))^{-} \Big] \end{aligned} \tag{4.39}$$

Considering all the road obstacles, choosing the lowest acceleration that results from any of them is necessarily a safe choice. In the same way, the minimum is also considered across time, predicting how the distance and speed error will evolve if the ego-vehicle keeps constant speed, using the parameter γ_{a} ($0 \leq \gamma_{\mathrm{a}} \leq 1$) to model the uncertainty about the future. The safety distance is computed using the obstacle's speed to avoid triggering oscillation when the ego-car slows down to match the leading speed, reducing the required safety speed which, in turn, might cause the car to speed up again and so forth.

Complete stop

Due to the design of the SLK policy, it's evident that a zero speed sample will not cause the car to stop at the required position. To compensate for this behavior and enable a precise stop at a given location, the additional complete stop policy is added to the set. This policy is enabled by the presence of a null speed limit sample and iteratively computes the required constant acceleration to end the motion at the stop location.

The complete stop also takes into account another problem that arises when the stop location is derived from sensors. Due to the noise in the detection process, the stop coordinates may oscillate around the true value and this might cause oscillations in the brake set-point. Although this is usually not a safety concern, the hesitating brake action is very uncomfortable. To avoid this situation, the complete stop policy introduces an hysteresis to filter small variations and iteratively keeps the minimum among the computed accelerations.

Assuming s_0 is the car's curvilinear abscissa, v_{car} the current car speed, s_S is the stop's coordinate, and $a_{\mathrm{CS},i-1}$ the previous acceleration value, the acceleration set-point is computed as:

$$a_{\mathrm{CS},i} = \min \left\{ a_{\mathrm{CS},i-1} \, , \, \frac{v_{car}^2}{2(s_S - s_0)} \right\} \tag{4.40}$$

The hysteresis is reset is the different between the set-points is higher than a certain threshold and when the policy is no more applicable.

Integration scheme

A proper integration scheme is needed to compute the final reference $a(t)$ from the policies set. A popular approach in robotics is to define some scenario-dependent priorities and apply a winner-take-all strategy [110]. This exclusive selection technique, applied to speed policy arbitration, makes hard to handle mixed situations like performing ACC while stopping at a junction.

To achieve a smooth and safe integration of all the policies, a different scheme is adopted: all the applicable policies are evaluated and the acceleration is selected to be the lowest among all. This approach seamlessly handles mixed scenarios and, because all the constraints specify an *upper bound* for speed, choosing the lowest acceleration always result in a safe and feasible choice.

The applicability conditions for the speed policies are:

Speed limit keeping : always applicable.

Adaptive cruise control : presence of leading road obstacles in the road.

Complete stop : presence of a 0 value in $v_{\max}$.

Results are presented in the next section.

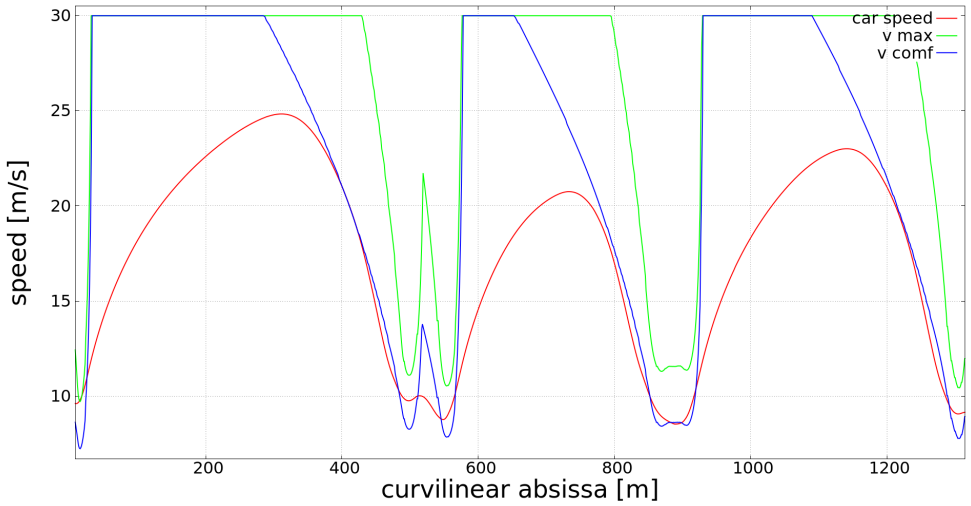
4.4.3 Results

The reactive speed policies successfully provide a consistent smooth acceleration reference for the driving cases of interest. The results are presented from the simulation of scenarios designed to show the behavior of the three developed policies.

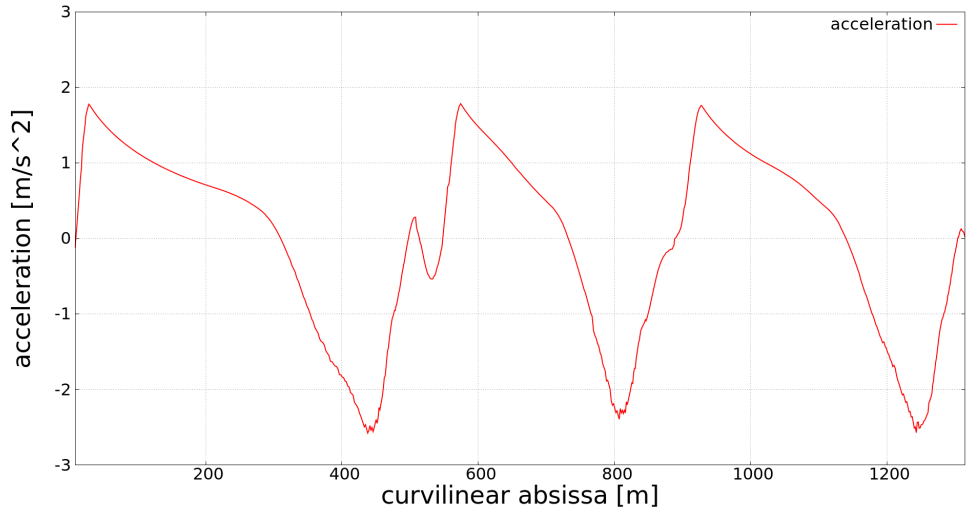
Figure 4.25a shows the car speed against the speed limits. The speed limit is saturated at $30m/s$ and the low peaks are due to the lateral acceleration constraints. The car's speed is always lower than $v_{\max}$, loosely following v_{comf} but allowing some violations to avoid excessive braking action. Image 4.25b depicts the relative acceleration reference showing that it changes smoothly and thus results in a comfortable driving.

To evaluate the adaptive cruise control policy, figure 4.26 shows the car's reaction to the presence of a road obstacle. The obstacle has a constant speed of $20m/s$ and is detected at a distance of about $40m$. Note how the car's speed, despite a speed limit of $50m/s$, adjust to the obstacle's speed and the distance from the obstacle sets on the intended value of $20m$ without oscillations. Another case is provided in picture 4.27 that corresponds to a late detection of a slow obstacle. This scenario is more challenging and requires a more aggressive reaction. The obstacle's speed is $10m/s$ and the detection distance about $20m$. Moreover, after few seconds, the obstacle suddenly brakes with acceleration $-3m/s^2$ to a complete stop, forcing the car to another emergency brake. Note how the car's braking action is firm to avoid a potential collision.

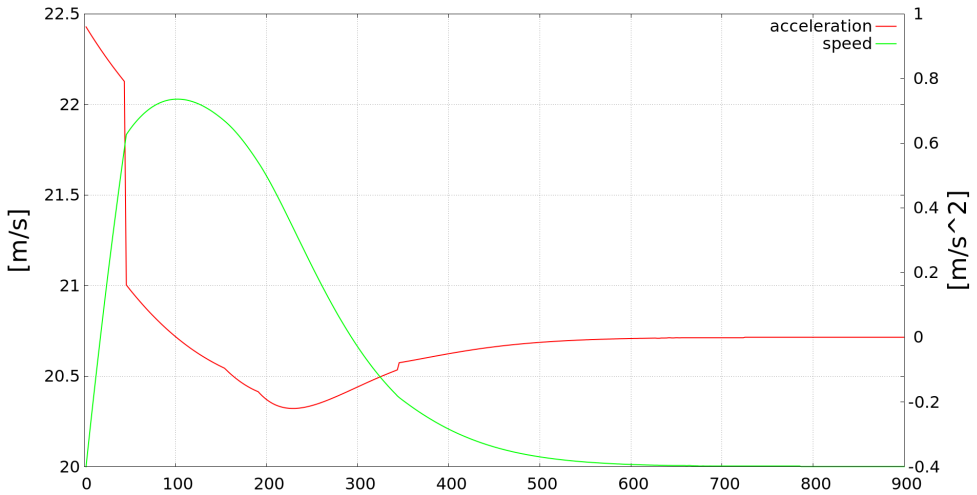
Finally, figure 4.28 shows the acceleration reference deriving from the complete stop policy when approaching a stop line. The car end it's motion at the intended distance from the stop line.



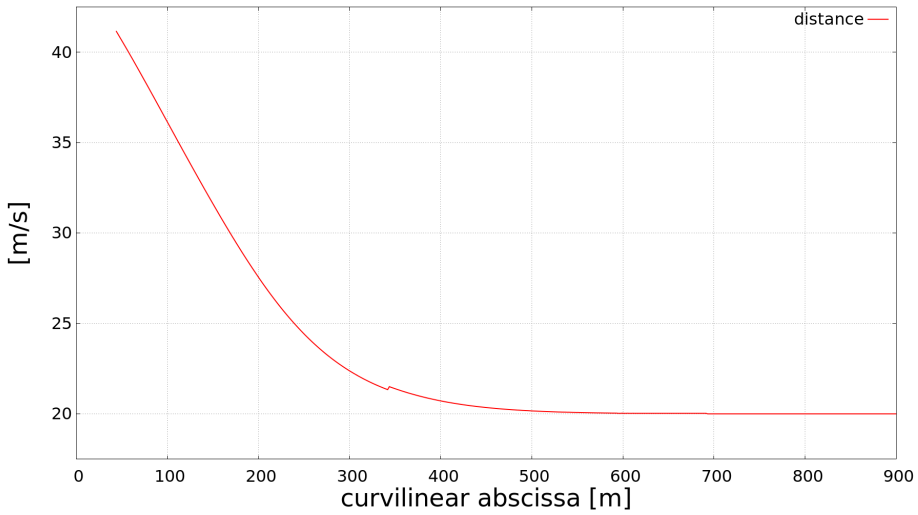
(a) Speed of the car and speed limits.



(b) Acceleration reference from the speed limit keeping policy.



(a) Speed and acceleration of the ego-car.



(b) Distance from the leading vehicle.

Figure 4.26: ACC results when approaching a vehicle from the distance.

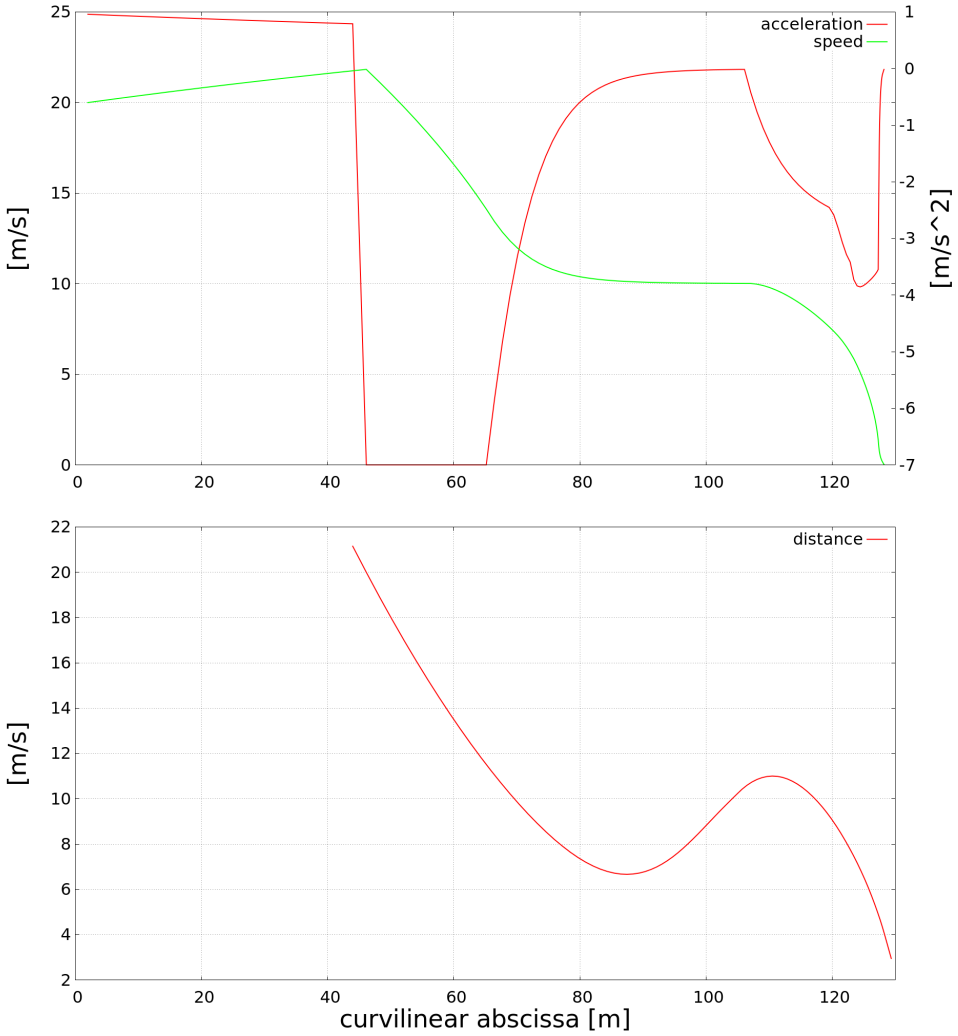


Figure 4.27: ACC results when a slow obstacle is detected at short distance.

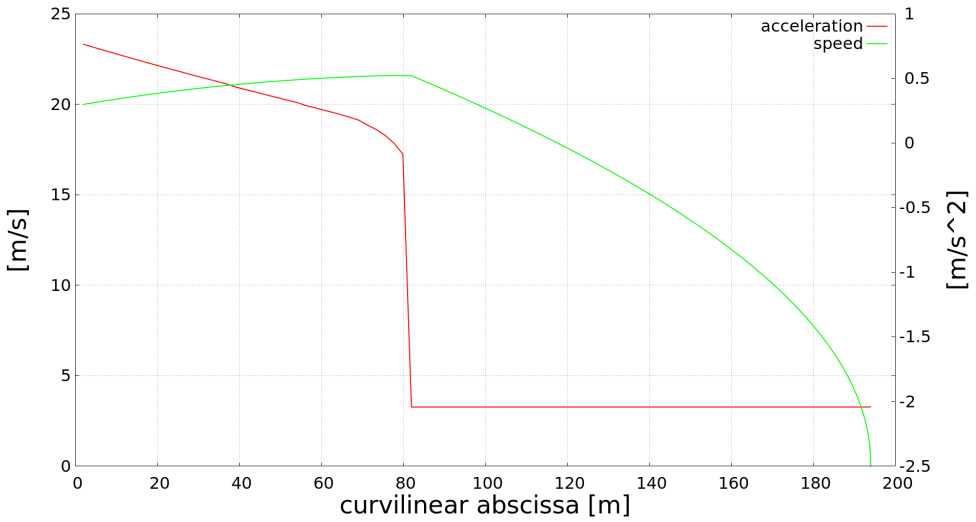


Figure 4.28: Complete stop.

4.5 Trajectory planning results

The presented algorithms to path and speed planning are integrated to produce a local trajectory for the car. First, the path planning routine is executed to find a feasible path that keeps the car within the driving reference and avoids static obstacles. The final path is thoroughly checked, providing for each sample the distance from the navigation borders and the closest obstacle. These informations are used to adjust the speed accordingly: the speed limit is set to 0 before a collision and lowered when passing close to obstacles and borders. Secondly, an acceleration reference is computed in order to satisfy the speed's upper bounds, safety distance and possible active stop lines. Combining path and acceleration, the trajectory for the car is fully specified.

As previously stated, the separate solution of path planning and speed tuning makes the overall problem easier, but the solution found is potentially sub-optimal. The most relevant example of sub-optimality is the behavior of the car when facing a road obstacle that cannot be avoided with the pure braking action. This undesirable situation can arise from detection latencies and pose a serious safety threat. To deal with this situation and other potential unwanted behaviors, a dedicated preprocessing is carried out before the planning phase. Each road obstacle is checked to establish whether it can be avoided with the pure braking action. Accounting for delays and a fixed reaction time, the motion of the car subject to an emergency brake is compared with the future states of the obstacles to see if a collision state is predicted. If the obstacle results in a potential collision, then it's not treated just like a road obstacle, but it's instantaneous state is used also as a raw obstacle, triggering an avoidance maneuver. In this way, beside the aggressive braking, the car also tries a swerve to avoid the impact. Figure 4.29 shows a sequence in which the car avoids a collision in this way; the potential is provided to emphasize that the road obstacle is treated also like a raw obstacle.

Another unwanted behavior is the car trying to avoid an obstacle that totally occludes the navigation reference. While a collision is avoided adjusting

the speed limit, the steering action causes a very uncomfortable driving feeling. To avoid this minor issue, the raw obstacles that leave a free passage that is not enough for the car to comfortably fit, are treated as static road obstacles (subject to previous check). Using this trick, the car comes comfortably to a stop without modifying the regular driving path.

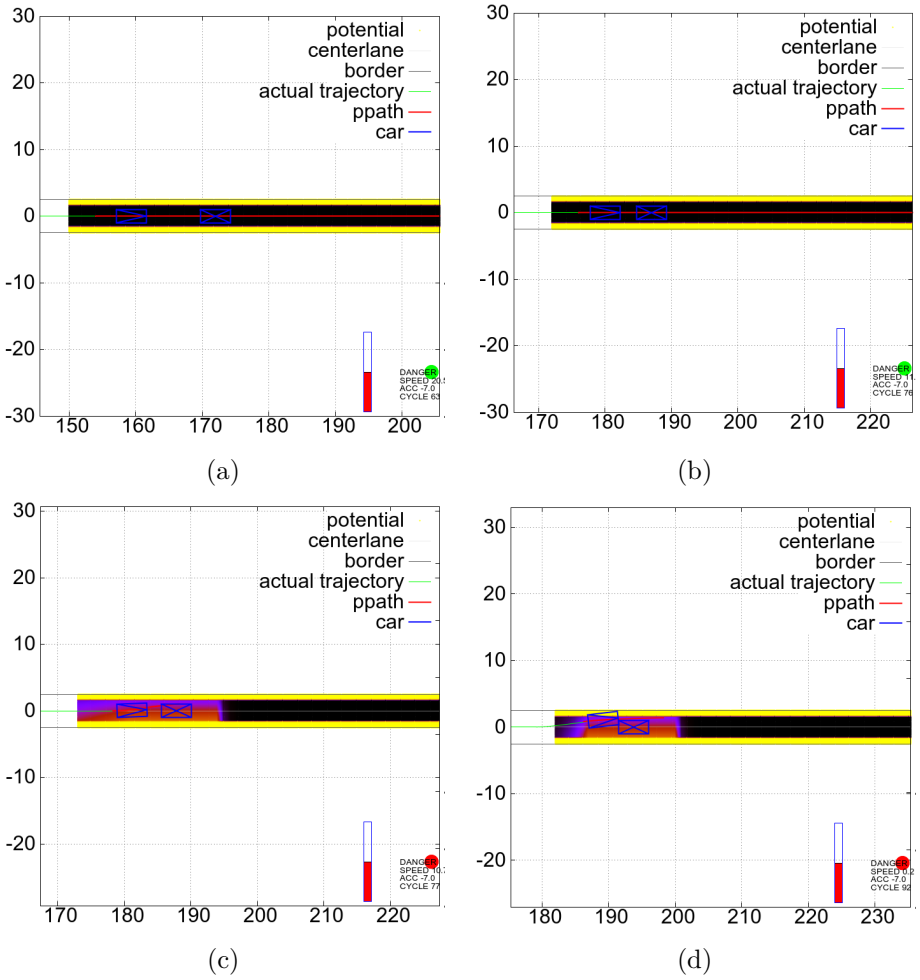


Figure 4.29: Sequence showing an emergency avoidance. A close, slow obstacle is detected triggering a full brake 4.29a. The car comes close to the obstacle within a safe distance 4.29b, ,but the obstacle suddenly brakes 4.29c. The car tries an avoidance maneuver to mitigate the impact 4.29d.

4.5.1 Conclusion and future development

The described local trajectory planner is able to provide a feasible, collision-free local trajectory to execute a wide range of maneuvers like lane keeping, lane change and basic intersection negotiation. Leveraging a separate solution method for path and speed, it can deliver real-time results with limited computational effort. The innovative cost function design departs from the classic quadratic cost to deal with problems arising from real driving cases and often neglected in the purely theoretic solution approaches.

The proposed method has, however, some limitations that should be overcome.

First, the separate solution method that permits real-time performance is also a weak spot of the algorithm. Planning the speed having the path already fixed leads to possible sub-optimality in the resulting behavior. Fixing the most relevant unwanted behaviors requires hand-designed "tricks" that, however, fail to be general and cover other cases. In particular, complex configurations of moving obstacles can only be handled in a conservative way with braking action. Resolving this issue requires the simultaneous planning of path and speed. To perform this joint planning efficiently, the algorithm might use a series of motion approximations with increasing detail in order to perform the final complete trajectory optimization in a well defined spacial-temporal region.

The other major limitation is related to the speed tuning method. This simple solution approach has the drawback of handling only upper bounds constraints on the speed. This limitation is due to the chosen integration scheme and restricts the planner from handling complex merging scenario where the car might be required to have a minimum speed in order to safely merge into the traffic. This restriction is not severe and can be solved introducing a new policy, activated in mutual exclusion with the others, explicitly designed to handle such case. Such problem would not arise with a joint solution method.

Chapter 5

Conclusions And Future Development

The previous chapter introduced some key components, in the three main autonomous vehicles' functional areas, that enable autonomous navigation from sensory information in structured environments. In the perception domain, a novelty application of CNN to ego-lane detection was introduced, along with a semi-automatic annotation technique that enabled the fast and cheap development of a relatively large dataset. In the data fusion area, a new approach to road path estimation from multiple sensors was proposed. Using the moving window estimation technique, the algorithm achieved a good precision and satisfying stability despite the noise present in the input. Finally, in the planning domain, an efficient local trajectory planner, based on speed-tuning and variational methods, was introduced proving the ability to provide feasible, collision-free trajectory to implement several maneuvers.

The newly introduced components have been specifically designed and implemented to be employed on the autonomous driving platform DEEVA, described in section 1.5. Since the focus of this work is the practical application on a real vehicle, the development of all the algorithm needed to manage all the problematics arising from real-world applications. This research work tries

to be a point of contact between theory and application, aiming for a balance between theoretical justification and practical applicability.

To further prove the applicability of the outcomes of this thesis, the next section describes the setup and execution of a public demo, held in Parma in March 2017, which saw DEEVA driving in urban environment from sensory inputs with controlled traffics [111].

5.1 Demo: setup and execution

The platform used to demonstrate the autonomous driving ability is the DEEVA vehicle. The following sections detail the DEEVA's software setup and the overall demo structure.

5.1.1 DEEVA setup

DEEVA features a distributed software architecture that reflects the network of computing nodes. The communication is based on a publisher-subscriber protocol that establish the format for the data exchange. Exploiting the UPD broadcast, this protocol allows the transparent communication between any application regardless of their physical location.

A complete list and description of the algorithms used to enable level-3 autonomy is provided next.

Perception

The sensor suite used for the perception task is composed of solely the FFN stereo couple and the front RADAR, see 1.7 for the sensors' map. The FFN right camera is used to execute a lane detection algorithm, the CenterNET and a street sign detector. The stereo information is used by the barriers and curbs detection algorithm and by a stereo obstacle detector. The obstacle detector also integrates two classifiers to identify vehicles and pedestrians.

These perception algorithm are distributed on two separate PCs, one runs the lane detector, the CenterNET and the street sign detector, while the other takes care of curbs/barriers and obstacles detection.

Data fusion

The data-fusion module runs on a dedicated PC, communicating with both the perception algorithms and the planner/control. It is composed of 4 separate *filters* that accomplish different tasks.

The fundamental block is the positioning filter. Its information is used by all the other filters and it's dispatched to the planning module. This filter estimates the positioning of the vehicle in two different references systems: *global* and *navigation*. The global reference system provides the absolution localization of car using a low-cost GPS and matching the detected features with a geo-localized map. The global positioning represents the most likely estimate of the car's absolute location in the world, but lacks any local consistency in that it can discontinuously jump to follow new detections. The navigation reference, instead, derives from the integration of the car's odometry, refined with accelerations and angular rates. It has no absolute meaning, but provides local consistency.

The road path estimation filter provides a navigation reference from lane markings, CenterNET, RADAR and barriers/curbs as described in chapter 3.

The obstacle filter produces a refined estimation of the obstacle's position, motion and classification using the informations from the stereo obstacle detector, RADAR and pedestrian/vehicle classifier.

Finally, the map manager filter handles the map database, dispatching the relevant geo-localized features or recorded navigation tracks when available. The use of recorded tracks, which contradicts the scope of this work, is exploited to handle the roundabouts present in the course without interrupting the autonomous driving. Since the correct detection of a driving reference when in roundabouts is beyond the ability of this setup (due to both sensors' FOV and algorithmic limitations), when the car approaches a roundabout, the

system sends a recorded track to be used in place of the one derived from the sensors. The absolute localization is precise enough to allow navigation from maps because the beginning of roundabouts is very rich in features (street signs, curbs, markings).

Planning

The planning module uses another dedicated PC and integrates a basic behavioral layer with the local trajectory planner previously introduced. Since the demo follows a predetermined course, no routing algorithm is needed.

The behavioral planner is responsible for providing basic intersection handling, yielding to pedestrian and regular lane navigation. The handling of intersections (roundabout) is based on simple occupation considerations: if a vehicle is detected within the intersection limits, the ego-car will stop and yield, navigating the recorded intersection track when free. Pedestrian are handled similarly, if they're detected near the zebra crossing, the stop line will activate causing the car to stop and wait for the area to be clear. For safety reason, pedestrian detected within the road boundaries always trigger a stop line before their position. This layer also takes care of switching between perceived and recorded track when approaching roundabouts.

The selected navigation reference, with possible active stop lines, is used by the local trajectory planner to find a proper trajectory for the car.

The same computing node hosts the low level controller for the car. The control scheme is split between steering and gas/brake. A path tracker based on pure pursuit [112] controls the steering wheel. The gas and brake pedals are actuated from a couple of second order linear controllers that compensate the car dynamics and follow the speed and acceleration reference.

5.1.2 Demo structure

The public demonstration was held in Parma's neighborhood Parmamia, on March 23, 2017. The demo area was in a controlled traffic state, designed to

show all the autonomous features of the DEEVA's setup. Figure 5.1 shows the demo's course in red, highlighting different zones corresponding to specific challenges. Picture 5.2, provided by [111], shows DEEVA during the demo

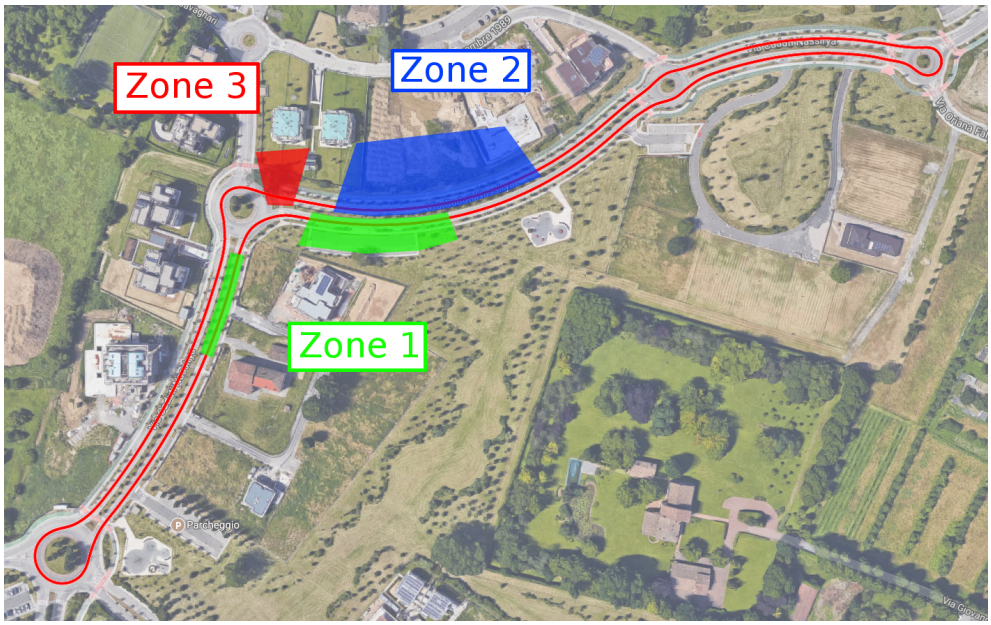


Figure 5.1: Demo course with specific zones highlighted.

execution.

In zone 1, parked vehicles were present on the side of the road, some were intentionally parked there other were already in place. Since the cars were actually inside the road boundaries, DEEVA demonstrated the ability to avoid static obstacles. Picture 5.4a shows a capture from the inside of the vehicle when avoiding a side obstacle.

Zone 2 is where the ACC ability was displayed, safely following one of VisLab's vehicle. This area ended with a simulated jam, causing a stop-and-go like behavior of DEEVA. Picture 5.4b shows the simulated jam.

Finally, in zone 3 the pedestrian crossing was activated by our actors and DEEVA stopped, waiting for the zebra to be clear. This can be seen in picture

Figure 5.2: DEEVA view during the demo.

Figure 5.3: Driver and front passenger monitoring the demo safe execution.

5.4c.

As already described, the car navigated using the sensory informations everywhere except when in roundabouts; those were negotiated using pre-recorded tracks and global localization coming from low cost GPS and feature matching. DEEVA successfully completed multiple runs, during which the driver and passenger were monitoring the execution for safety reasons, see figure 5.3 from [111].

(a) Zone 1: avoidance of parked vehicles.

(b) Zone 2: ACC and traffic jam.

(c) Zone 3: yield at pedestrian crossing.

Figure 5.4: Captures of the demo execution from inside the car. The PC that can be seen on the right was used for monitoring the health of the system.

5.2 Future development

The present work introduced important elements to enable autonomous driving from sensory input. For what concerns the detection and data-fusion, emphasis was placed on improving the stability and reliability of the derived navigation reference. Planning main focus was to provide a natural and comfortable driving and to guarantee robustness to noisy references.

The CNN based ego-lane detection can be further improved collecting more data and covering more cases. Moreover, the fixed-length limitation should be removed and the estimation of other lanes enabled. In it's ultimate evolution, this network should be able to identify all the available driving reference in the image, correctly identifying also the topology of intersections.

The road path estimation robustness could be improved using, when available, maps informations to exclude false detections and provide a prior on the road shape. Estimating intersections will eventually be a required capability of this component also.

Finally, the local trajectory planner should be evolved in order to manage general timing constraints on the vehicle states.This capability is required to perform complex merging maneuver, for example when moving "bubbles" in the traffic can be exploited to safely merge in.

As a general development direction, I think the focus should shift on empowering the feedback connections between the three functional ares. The classic "feed-forward" view of perception, data fusion and planning has a severe limitation: it fails to capture the "attention" mechanism likely needed to handle complex scenarios within human-level reaction times. An example of such feedback interaction could be: the behavioral planning communicates the intention of changing lane, in response the data-fusion changes the acceptance threshold of incoming obstacle from that lane, possibly allowing some false-positives (relatively harmless in this situation), but minimizing the false-negatives that would result in a dangerous situation. Moreover, as part of the feedback, the data fusion can ask the perception to focus on a particular region

that has high uncertainty. This feedback interaction is a proposal that moves toward an architecture that minimize the information loss due to the separation of the functionalities in different blocks. Ideally, the architecture should allow the computation to be conditioned on every relevant bit of information, regardless of its functional classification.

The achievement of level 5, full-autonomous driving seems to get closer by the day. The potential benefits of this revolutionary technology are huge and, whether it will be a product or a service (we will eventually see both applications), the impact will be profound. The methods proposed in this thesis provide only a fraction of the functionalities required for AD, but represent a small step toward its practical solution.

Bibliography

[1] Directorate General for Mobility European Commission and Transport. Road safety evolution in eu. Technical report, CARE, November 2016.

[2] NHTSA's National Center for Statistics and Analysis. Traffic safety facts 2015 data. Technical report, US Department of Transportation, June 2017.

[3] NHTSA. Critical reasons for crashes investigated in the national motor vehicle crash causation survey. Technical report, US Department of Transportation, February 2015.

[4] NHTSA. The road ahead, national highway traffic safety administration strategic plan, 2016–2020. Technical report, US Department of Transportation, October 2016.

[5] Susan M. Paddock Nidhi Kalra. Driving to safety. how many miles of driving would it take to demonstrate autonomous vehicle reliability? Technical report, RAND, February 2015. URL: `https://www.rand.org/pubs/research_reports/RR1478.html`.

[6] Raphael E. Stern, Shumo Cui, Maria Laura Delle Monache, Rahul Bhadani, Matt Bunting, Miles Churchill, Nathaniel Hamilton, R'mani Haulcy, Hannah Pohlmann, Fangyu Wu, Benedetto Piccoli, Benjamin Seibold, Jonathan Sprinkle, and Daniel B. Work. Dissipation of stop-and-go waves via control of autonomous vehicles: Field experiments.

CoRR, abs/1705.01693, 2017. URL: http://arxiv.org/abs/1705.01693.

[7] Dwight Hennessy and David Wiesenthal. Traffic congestion, driver stress, and driver aggression. 25:409 – 423, 01 1999.

[8] Americans spend an average of 17,600 minutes driving each year, September 2016. URL: http://newsroom.aaa.com/2016/09/americans-spend-average-17600-minutes-driving-year/.

[9] United State CENSUS. Commuting in the united states: 2009, 2011.

[10] Emilio Frazzoli. Perception for action: On nutonomy's vision for autonomous driving. July 2017.

[11] *Reinventing the Automobile: Personal Urban Mobility for the 21st Century*. MIT Press, 2010. URL: https://books.google.it/books?id=9d8xnQAACAAJ.

[12] Kevin Spieser, Kyle Treleaven, Rick Zhang, Emilio Frazzoli, Daniel Morton, and Marco Pavone. *Toward a Systematic Approach to the Design and Evaluation of Automated Mobility-on-Demand Systems: A Case Study in Singapore*, pages 229–245. Springer International Publishing, Cham, 2014. URL: https://doi.org/10.1007/978-3-319-05990-7_20, doi:10.1007/978-3-319-05990-7_20.

[13] Science: Radio auto. August 1926.

[14] "pahntomauto" to be operated here, June 1932. URL: https://news.google.com/newspapers?id=PthNAAAAIBAJ&sjid=yYoDAAAAIBAJ&pg=6442,3879017.

[15] Norman Bel Geddes. *Magic motorways*. [New York] Random house, 1940.

[16] Ernst Dieter Dickmanns and Volker Graefe. Dynamic monocular machine vision. *Machine Vision and Applications*, 1(4):223–240, Dec 1988. URL: https://doi.org/10.1007/BF01212361, doi:10.1007/BF01212361.

[17] Ernst D Dickmanns et al. Vehicles capable of dynamic vision. In *IJCAI*, volume 97, pages 1577–1592, 1997.

[18] Dean A. Pomerleau. Advances in neural information processing systems 1. chapter ALVINN: An Autonomous Land Vehicle in a Neural Network, pages 305–313. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989. URL: http://dl.acm.org/citation.cfm?id=89851.89891.

[19] the no hands across america, 1995. URL: http://www.cs.cmu.edu/~tjochem/nhaa/nhaa_home_page.html.

[20] A. Broggi. *Automatic Vehicle Guidance: The Experience of the ARGO Autonomous Vehicle*. World Scientific, 1999. URL: https://books.google.it/books?id=K1e55e8wiEUC.

[21] Q. Chen, U. Ozguner, and K. Redmill. Ohio state university at the 2004 darpa grand challenge: developing a completely autonomous vehicle. *IEEE Intelligent Systems*, 19(5):8–11, Sept 2004. doi:10.1109/MIS.2004.48.

[22] S. Thrun. Winning the darpa grand challenge: A robot race through the mojave desert. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, Sept 2006. doi:10.1109/ASE.2006.74.

[23] Urban chanllenge. URL: http://archive.darpa.mil/grandchallenge/.

[24] Massimo Bertozzi, Luca Bombini, Alberto Broggi, Michele Buzzoni, Elena Cardarelli, Stefano Cattani, Pietro Cerri, Stefano Debattisti, Rean Isabella Fedriga, Mirko Felisa, Luca Gatti, Andrea Giacomazzo, Paolo Grisleri, Maria Chiara Laghi, Luca Mazzei, Paolo Medici, Matteo Panciroli, Pier Paolo Porta, and Paolo Zani. The vislab intercontinental autonomous challenge: 13,000 km, 3 months, no driver. In *Procs. 17th World Congress on ITS*, October 2010.

[25] Alberto Broggi, Pietro Cerri, Stefano Debattisti, Maria Chiara Laghi, Paolo Medici, Daniele Molinari, Matteo Panciroli, and Antonio Prioletti. PROUD - public road urban driverless-car test. *IEEE Trans. Intelligent Transportation Systems*, 16(6):3508–3519, 2015. URL: `https://doi.org/10.1109/TITS.2015.2477556`, `doi:10.1109/TITS.2015.2477556`.

[26] Daniele Molinari. Proud: a planning approach based on potential map and speed tuning method.

[27] Julius Ziegler, Philipp Bender, Markus Schreiber, Henning Lategahn, Tobias Strauss, Christoph Stiller, Thao Dang, Uwe Franke, Nils Appenrodt, Christoph Gustav Keller, Eberhard Kaus, Ralf G. Herrtwich, Clemens Rabe, David Pfeiffer, Frank Lindner, Fridtjof Stein, Friedrich Erbs, Markus Enzweiler, Carsten Knöppel, Jochen Hipp, Martin Haueis, Maximilian Trepte, Carsten Brenk, Andreas Tamke, Mohammad Ghanaat, Markus Braun, Armin Joos, Hans Fritz, Horst Mock, Martin Hein, and Eberhard Zeeb. Making bertha drive - an autonomous journey on a historic route. *IEEE Intell. Transport. Syst. Mag.*, 6(2):8–20, 2014.

[28] URL: `https://waymo.com/`.

[29] 2015. URL: `http://www.autonews.com/article/20151014/OEM06/151019938/tesla-beams-down-autopilot-mode-to-model-s`.

[30] SAE International. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. Technical report, SAE, September 2016. URL: standards.sae.org/j3016_201609/.

[31] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[32] Scott Drew Pendleton, Hans Andersen, Xinxin Du, Xiaotong Shen, Malika Meghjani, You Hong Eng, Daniela Rus, and Marcelo H. Ang. Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5(1), 2017. URL: http://www.mdpi.com/2075-1702/5/1/6, doi:10.3390/machines5010006.

[33] Sagar Behere and Martin Törngren. A functional architecture for autonomous driving. In *Proceedings of the First International Workshop on Automotive Software Architecture*, WASA '15, pages 3–10, New York, NY, USA, 2015. ACM. URL: http://doi.acm.org/10.1145/2752489.2752491, doi:10.1145/2752489.2752491.

[34] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. URL: http://arxiv.org/abs/1604.07316.

[35] K. Ito, K. Hashimoto, and Y. Shibata. V2x communication system for sharing road alert information using cognitive network. In *2017 IEEE 8th International Conference on Awareness Science and Technology (iCAST)*, pages 533–538, Nov 2017. doi:10.1109/ICAwST.2017.8256515.

[36] A. Ometov and S. Bezzateev. Multi-factor authentication: A survey and challenges in v2x applications. In *2017 9th International Congress*

on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), pages 129–136, Nov 2017. doi:10.1109/ICUMT.2017.8255200.

[37] URL: http://nutonomy.com/.

[38] URL: https://www.uber.com/info/atg/.

[39] A. Broggi, S. Debattisti, P. Grisleri, and M. Panciroli. The deeva autonomous vehicle platform. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 692–699, June 2015. doi:10.1109/IVS.2015.7225765.

[40] Robert P. Loce, Raja Bala, and Mohan Trivedi. *Lane Detection and Tracking Problems in Lane Departure Warning Systems*, pages 432–. Wiley-IEEE Press, 2017. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7906248, doi:10.1002/9781118971666.ch11.

[41] Junyung Lee, Jaewoong Choi, Kyongsu Yi, Minyong Shin, and Bongchul Ko. Lane-keeping assistance control algorithm using differential braking to prevent unintended lane departures. *Control Engineering Practice*, 23:1 – 13, 2014. URL: http://www.sciencedirect.com/science/article/pii/S0967066113001901, doi:http://dx.doi.org/10.1016/j.conengprac.2013.10.008.

[42] S. J. Huang and C. C. Tsai. Vehicle lane detection and following based on vision system and laser scanner. In *2017 International Conference on Applied System Innovation (ICASI)*, pages 1446–1449, May 2017. doi:10.1109/ICASI.2017.7988188.

[43] Albert S. Huang, David Moore, Matthew Antone, Edwin Olson, and Seth Teller. Finding multiple lanes in urban road networks with vision and lidar. *Autonomous Robots*, 26(2):103–122, Apr 2009. URL: https://doi.org/10.1007/s10514-009-9113-3, doi:10.1007/s10514-009-9113-3.

[44] S. Kammel and B. Pitzer. Lidar-based lane marker detection and mapping. In *2008 IEEE Intelligent Vehicles Symposium*, pages 1137–1142, June 2008. doi:10.1109/IVS.2008.4621318.

[45] M. Felisa and P. Zani. Robust monocular lane detection in urban environments. In *2010 IEEE Intelligent Vehicles Symposium*, pages 591–596, June 2010. doi:10.1109/IVS.2010.5548028.

[46] Hanspeter A. Mallot, H. H. Bülthoff, J. J. Little, and S. Bohrer. Inverse perspective mapping simplifies optical flow computation and obstacle detection. *Biological Cybernetics*, 64(3):177–185, Jan 1991. URL: https://doi.org/10.1007/BF00201978, doi:10.1007/BF00201978.

[47] S. Nedevschi, F. Oniga, R. Danescu, T. Graf, and R. Schmidt. Increased accuracy stereo approach for 3d lane detection. In *2006 IEEE Intelligent Vehicles Symposium*, pages 42–49, 2006. doi:10.1109/IVS.2006.1689603.

[48] W. Song, M. Fu, Y. Yang, M. Wang, X. Wang, and A. Kornhauser. Real-time lane detection and forward collision warning system based on stereo vision. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 493–498, June 2017. doi:10.1109/IVS.2017.7995766.

[49] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[50] Yann LeCun, Koray Kavukcuoglu, and ClÃĺment Farabet. *Convolutional networks and applications in vision*, pages 253–256. 2010. doi:10.1109/ISCAS.2010.5537907.

[51] Seong-Woo Park and Ki-Sang Hong. Practical ways to calculate camera lens distortion for real-time camera calibration. *Pattern Recognition*, 34(6):1199 – 1206, 2001. URL: http://www.sciencedirect.com/science/article/pii/S0031320300000686, doi:http://dx.doi.org/10.1016/S0031-3203(00)00068-6.

[52] M. Bertozzi, L. Bombini, A. Broggi, P. Zani, P. Cerri, P. Grisleri, and P. Medici. Gold: A framework for developing intelligent-vehicle vision applications. *IEEE Intelligent Systems*, 23(1):69–71, Jan 2008. doi: 10.1109/MIS.2008.6.

[53] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[54] E.J. Wilczynski. *Projective differential geometry of curves and ruled surfaces*. B.G. Teubners Sammlung von Lehrbüchern auf dem Gebiete der mathematischen Wissenschaften mit Einschluss ihrer Anwendungen. B.G. Teubner, 1921. URL: https://books.google.it/books?id=LEpLAAAAMAAJ.

[55] Elementi di analisi per visione artificiale. URL: http://www.ce.unipr.it/~medici/geometry.pdf.

[56] R.M. Rogers. *Applied Mathematics in Integrated Navigation Systems*. Number v. 1 in AIAA education series. American Institute of Aeronautics and Astronautics, 2003. URL: https://books.google.it/books?id=dfS2WcYba9wC.

[57] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL: http://arxiv.org/abs/1409.1556.

[58] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL: http://arxiv.org/abs/1207.0580.

[59] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL: http://arxiv.org/abs/1502.03167.

[60] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL: `http://arxiv.org/abs/1412.6980`.

[61] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer, 2012. URL: `https://doi.org/10.1007/978-3-642-24797-2`, `doi:10.1007/978-3-642-24797-2`.

[62] O. Vinyals, S. Bengio, and M. Kudlur. Order Matters: Sequence to sequence for sets. *ArXiv e-prints*, November 2015. `arXiv:1511.06391`.

[63] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014. URL: `http://arxiv.org/abs/1411.4038`.

[64] Shuliang Zhu, Jianqiang Wang, Tao Yu, and Jiao Wang. A method of lane detection and tracking for expressway based on ransac. In *2017 2nd International Conference on Image, Vision and Computing (ICIVC)*, pages 62–66, June 2017. `doi:10.1109/ICIVC.2017.7984519`.

[65] S. Nedevschi, R. Schmidt, T. Graf, R. Danescu, D. Frentiu, T. Marita, F. Oniga, and C. Pocol. 3d lane detection system based on stereovision. In *Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No.04TH8749)*, pages 161–166, Oct 2004. `doi:10.1109/ITSC.2004.1398890`.

[66] Chris Kreucher, Sridhar Lakshmanan, and Karl Kluge. A driver warning system based on the lois lane detection algorithm. In *Proceedings of IEEE International Conference on Intelligent Vehicles*, volume 1, pages 17–22. Stuttgart, Germany, 1998.

[67] Camillo J Taylor, Jana Košecká, Robert Blasi, and Jitendra Malik. A comparative study of vision-based lateral control strategies for au-

tonomous highway driving. *The International Journal of Robotics Research*, 18(5):442–453, 1999.

[68] Nicholas Apostoloff and Alexander Zelinsky. Robust vision based lane tracking using multiple cues and particle filtering. In *Intelligent Vehicles Symposium, 2003. Proceedings. IEEE*, pages 558–563. IEEE, 2003.

[69] Johannes Rabe, Martin Hübner, Marc Necker, and Christoph Stiller. Ego-lane estimation for downtown lane-level navigation. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 1152–1157. IEEE, 2017.

[70] Albert S Huang, David Moore, Matthew Antone, Edwin Olson, and Seth Teller. Finding multiple lanes in urban road networks with vision and lidar. *Autonomous Robots*, 26(2):103–122, 2009.

[71] Soren Kammel and Benjamin Pitzer. Lidar-based lane marker detection and mapping. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 1137–1142. IEEE, 2008.

[72] Lars B Cremean and Richard M Murray. Model-based estimation of off-highway road geometry using single-axis ladar and inertial sensing. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1661–1666. IEEE, 2006.

[73] Jorge Hernández and Beatriz Marcotegui. Filtering of artifacts and pavement segmentation from mobile lidar data. In *ISPRS Workshop Laserscanning 2009*, 2009.

[74] Kesav Kaliyaperumal, Sridhar Lakshmanan, and Karl Kluge. An algorithm for detecting roads and obstacles in radar images. *IEEE Transactions on Vehicular Technology*, 50(1):170–182, 2001.

[75] Bing Ma, Sridhar Lakshmanan, and Alfred O Hero. Simultaneous detection of lane and pavement boundaries using model-based multisensor fusion. *IEEE Transactions on Intelligent Transportation Systems*, 1(3):135–147, 2000.

[76] Albert S Huang. Lane estimation for autonomous vehicles using vision and lidar. 2010.

[77] James B Rawlings. Moving horizon estimation. *Encyclopedia of Systems and Control*, pages 1–7, 2013.

[78] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[79] Dmitrii Konstantinovich Faddeev and Vera Nikolaevna Faddeeva. Computational methods of linear algebra. 2007.

[80] W Zeng and RL Church. Finding shortest paths on real road networks: the case for a. *International journal of geographical information science*, 23(4):531–543, 2009.

[81] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[82] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012.

[83] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: autonomous vehicles in city traffic*, volume 56. springer, 2009.

[84] Sebastian Brechtel, Tobias Gindele, and Rüdiger Dillmann. Probabilistic mdp-behavior planning for cars. In *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*, pages 1537–1542. IEEE, 2011.

[85] Simon Ulbrich and Markus Maurer. Probabilistic online pomdp decision making for lane changes in fully automated driving. In *Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on*, pages 2063–2067. IEEE, 2013.

[86] Alessandro De Luca, Giuseppe Oriolo, and Claude Samson. Feedback control of a nonholonomic car-like robot. *Robot motion planning and control*, pages 171–253, 1998.

[87] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

[88] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[89] Julius Ziegler and Christoph Stiller. Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 1879–1884. IEEE, 2009.

[90] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[91] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[92] Alonzo Kelly and Bryan Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. *The International Journal of Robotics Research*, 22(7-8):583–601, 2003.

[93] Julius Ziegler, Philipp Bender, Markus Schreiber, Henning Lategahn, Tobias Strauss, Christoph Stiller, Thao Dang, Uwe Franke, Nils Appenrodt, Christoph G Keller, et al. Making bertha driveâĂŤan autonomous journey on a historic route. *IEEE Intelligent Transportation Systems Magazine*, 6(2):8–20, 2014.

[94] Michael Otte and Emilio Frazzoli. {\mathrm {RRTˆ{X}}}: Real-time motion planning/replanning for environments with unpredictable obstacles. In *Algorithmic Foundations of Robotics XI*, pages 461–478. Springer, 2015.

[95] Eamonn Keogh and Abdullah Mueen. Curse of dimensionality. In *Encyclopedia of Machine Learning*, pages 257–258. Springer, 2011.

[96] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[97] Victor Guillemin and Alan Pollack. *Differential topology*, volume 370. American Mathematical Soc., 2010.

[98] John R Dormand and Peter J Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

[99] Ahmet Sami KILINÇ and Tamer Baybura. Determination of minimum horizontal curve radius used in the design of transportation structures, depending on the limit value of comfort criterion lateral jerk. *TS06G-Engineering Surveying, Machine Control and Guidance, Rome, Italy*, 2012.

[100] Philip E Gill, Walter Murray, and Margaret H Wright. Practical optimization. 1981.

[101] Richard L Burden and J Douglas Faires. Numerical analysis. 2001. *Brooks/Cole, USA*, 2001.

[102] P Kaelo and MM Ali. Some variants of the controlled random search algorithm for global optimization. *Journal of optimization theory and applications*, 130(2):253–264, 2006.

[103] Wyn L. Price. A controlled random search procedure for global optimisation. *The Computer Journal*, 20(4):367–370, 1977.

[104] Donald M Olsson and Lloyd S Nelson. The nelder-mead simplex procedure for function minimization. *Technometrics*, 17(1):45–51, 1975.

[105] Kaj Madsen, Serguei Zertchaninov, and Antanas Zilinskas. Global optimization using branch-and-bound. *Submitted to Global Optimization*, 1998.

[106] S Gudmundsson. *Parallel global optimization*. PhD thesis, M. Sc. Thesis, IMM, Technical University of Denmark, 1998.

[107] Roger Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970.

[108] Thomas D Gillespie. Vehicle dynamics. *Warren dale*, 1997.

[109] Hans Pacejka. *Tire and vehicle dynamics*. Elsevier, 2005.

[110] Joseph L Jones. *Robot programming: a practical guide to behavior-based robotics*. McGraw Hill, 2004.

[111] Vislab avvia i test su strade pubbliche a parma. URL: `https://www.quattroruote.it/news/nuove_tecnologie/2017/03/24/auto_autonome_vislab_avvia_i_test_su_strade_pubbliche_a_parma.html`.

[112] Omead Amidi and Chuck E Thorpe. Integrated mobile robot control. In *Mobile Robots V*, volume 1388, pages 504–524. International Society for Optics and Photonics, 1991.

Acknowledgments

I would like to start with the formal thanks. Thank to Gabriele, Francesco for helping me getting my PhD experience in this piece of paper (or .pdf, depending on the format you're reading). Thank to VisLab for making this work possible: having a real vehicle to support my researches made them much more interesting and closer to reality.

Working on the car is a real fun, but it's not always easy, this is why a big thank goes to my colleagues-teammate-friends Francesco, Andrea, Matteo, Gabriele and Antonio (the order has been determined with `cat names.txt | shuf`). Together, getting through the endless hours of testing and debugging was a little less scary. We shared many special moments like the firsts autonomous steps of DEEVA, the first successful high speed obstacle avoidance, the first time driving at 120Km/h in autonomous mode and the first autonomous drifting (everything was done in the name of science). They were always ready to have a good laugh through good[1], bad[2] and regular[3] times. Together even being seized by armed policemen, regularly for 3 days, was less scaring (anyway it showed that we still have a long way to go before people will accept autonomous vehicles without alerting the police forces). Thank you all. (ci vediamo dopo dai Cinesi)

I can't help but thanking also all my friends outside the work environment.

[1] Good time is when the new cutting-edge feature works right away... ok, we didn't have that many good times.

[2] Bad time is when nothing works and you have no idea why.

[3] Regular time is when nothing works but you know why.

You're too many to name you all, but you'll know its you. Thank you because the time together kept me from loosing my sanity in the intricate mazes of my work's problem, and staying sane is definitively important... unless you live in a smooth manifold $\mathbb{R}^3 \times \mathbb{S}^1$, because Nietzsche says that time is cyclical, and you are riding a cow with autonomous rain tyres...

A huge thank to my family, Monica, Giovanni and Luca, because you never stopped supporting me. I'm grateful to you all, today I am what I am because of your support, care and love.

Last but not least, from the very bottom of my heart, a sincere thank to my soon-to-be wife Ksenia, my endless source of joy and inspiration. The impact you have in my life, day by day, is absolutely invaluable.