Karlsruhe Institute of Technology

FZI

Integrating Combinatorial Reasoning and Continuous Methods for Optimal Motion Planning of Autonomous Vehicles

Master's Thesis
of

Ömer Şahin Taş

KIT – Department of Measurement and Control
FZI – Mobile Perception Systems

Reviewer:	Prof. Dr.-Ing. Christoph Stiller
Advisor:	Dipl.-Ing. Philipp Bender
Second Advisor:	Dipl.-Inform. Julius Ziegler

Work period: 15. March 2014 — 15. September 2014

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association
FZI – Research Center for Information Technology

Decleration

I hereby declare that I have developed and written the enclosed thesis completely by myself, and have only referred to the sources or means presented in bibliography.

Karlsruhe, September 15, 2014.

Abstract

Driving involves deciding among numerous combinatorial options encountered in traffic. These combinations arise due to presence of, among other things, traffic participants, static obstacles, and traffic lights. In real world scenarios optimal motion is attained only in selecting the most favorable of the combinatorial alternatives. For autonomous vehicles, this necessitates integrating combinatorial reasoning into motion planning. This work addresses that task by starting with an initial discussion on alternative methods for optimal motion planning. Once it is clarified that only continuous methods can yield truly optimal results, methods for extraction and evaluation of all of the feasible combinations are presented. As a simplifying assumption, the set of all obstacles is assumed to be perfectly known, *i.e.* sensor uncertainty is neglected. Moreover, overtaking is prohibited. To restrain the planned motion into a selected combination, necessary constraints on the planner are formalized. The formulations are done for both complete and path-velocity decomposed planning. Subsequently, the presented approaches are simulated on several non-signalized intersection scenarios. The latter were created by utilizing an existing geographical mapping framework. Finally with the help of the results obtained, the plannings are compared and evaluated.

Keywords: Motion planning, combinatorial reasoning, autonomous driving, continuous methods, nonlinear optimization, path-velocity decomposition.

Kurzfassung

Beim Fahren müssen Entscheidungen aus zahlreichen kombinatorischen Auswahlmöglichkeiten, die im Straßenverkehr vorkommen, getroffen werden. Diese Kombinationen entstehen typischerweise durch andere Verkehrsteilnehmer, statische Hindernisse und Verkehrsampeln. In der realen Welt, kann die optimale Bewegung nur dann erreicht werden, wenn die günstigste Kombination aus den Alternativen ausgewählt wird. Für autonome Fahrzeuge erfordert dies die Integration der kombinatorischen Schlussfolgerung in die Bewegungsplanung, was als ein wesentliches Problem identifiziert wird. Diese Arbeit geht auf das Problem zuerst durch eine Diskussion über alternative Methoden für optimale Bewegungsplanung ein. Nachdem herausgearbeitet wurde, dass nur kontinuierlichen Methoden tatsächlich optimale Ergebnisse liefern, werden Verfahren der Extraktion und Evaluation für alle möglichen Kombinationen vorgestellt. Als vereinfachende Annahme wird die Menge der Hindernisse als genau bekannt angenommen, d.h. Messunsicherheiten werden vernachlässigt und zusätzlich dazu wird Überholen untersagt. Um die geplante Bewegung innerhalb einer ausgewählten Kombination einschränken zu können, werden Bedingungen für den Bewegungsplaner formalisiert. Die Formulierungen werden sowohl für komplette als auch für Pfad-Geschwindigkeit zerlegte Planungen vorgestellt. Infolgedessen werden die dargestellten Ansätze auf mehreren nicht-signalisierten Kreuzungsszenarien simuliert, welche mit Hilfe eines vorhandenen geographischen Kartierungs-Frameworks erzeugt wurden. Die ermittelten Ergebnisse aus beiden Planungen werden verglichen und ausgewertet.

Schlagwörter: Bewegungsplanung, kombinatorische Schlussfolgerung, autonomes Fahren, kontinuierliche Methoden, nichtlineare Optimierung, Pfad-Geschwindigkeit Zerlegung.

Acknowledgements

I am very happy to have completed my master's thesis. Before I started the work, I was familiar with neither nonlinear optimization nor motion planning. Hence, from the beginning the thesis was a substantial challenge for me. In the end, I finished a program exceeding 6000 lines of code. Of course, like in any undertaking, the work I have accomplished fundamentally relied on mental advancement I gained throughout the years starting from childhood with family and subsequently continuing with friends, teachers and professors, with whom I have become acquainted. I express my sincere thanks to all of them.

Several people specifically have spent their valuable time by supporting me. First and foremost, I would like to express my very great appreciation to Mr. Philipp Bender; not only for his support, useful critiques and efforts while he was mentoring, but also for his patient guidance with programming. Secondly, I would like offer my special thanks to Mr. Julius Ziegler. His advice has been a great help in understanding the vehicle model and control related aspects. I would also like to extend my thanks to my friends who have supported me. Particularly, I would like to thank to Mr. Ömer Kehri and Mr. Emre Taşpolatoğlu for their friendly support.

The research presented in this thesis has been performed at the FZI Research Center for Information Technology, at the department on Mobile Perception Systems supervised by Prof. Dr.-Ing. Christoph Stiller. I am very grateful for working at an institute dedicated to realizing fully autonomous driving and which provides a perfect environment and convivial atmosphere for research.

My master's studies at the Karlsruhe Institute of Technology were financed by the German Academic Exchange Service (DAAD) and the Turkish Education Foundation (TEV). I would like to thank both organizations for trusting in me and bestowing me with this great opportunity.

Finally, I wish to thank my family for their support and encouragement throughout my studies.

Ömer Şahin Taş

Karlsruhe, September 15, 2014

Contents

List of Figures

List of Tables

Notations

Abbreviations

ABS	Anti-Lock Braking System
ACC	Adaptive Cruise Control
ADAS	Advanced Driver Assistant Systems
BFGS	Broyden Fletcher Goldfarb and Shanno Algorithm
COBYLA	Constrained Optimization by Linear Approximations
CP	Complete Planning
DARPA	Defense Advanced Research Projects Agency
ESP	Electronic Stability Program
EST	Expansive-Space Tree planner
GUI	Graphical User Interface
JOSM	Java OpenStreetMap Editor
KKT	Karush Kuhn Tucker Conditions
MMA	Method of Moving Asymptotes
NHTSA	National Highway Traffic Safety Administration
NLP	Nonlinear Programming
OSM	OpenStreetMap
PATH	Program on Advanced Technology for the Highway
PRM	Probabilistic Roadmap Planner
PT	Path-Time
PVD	Path-Velocity Decomposition
RRT	Rapidly-exploring Random Tree planner
SLSQP	Sequential Least Squares Programming
SQP	Sequential Quadratic Programming
SRT	Sampling-Based Roadmap of Trees
TCS	Traction Control System
WHO	World Health Organization

Acronyms

$\boldsymbol{\alpha}_k$	step width at k^{th} step
$a_{\max}$	acceleration limit
$\mathbf{B}_k$	approximated Hessian at k^{th} step
b_{sign}	sign of a binary combination
$\mathbf{c}$	equality constraints obtained from KKT conditions
C	convex set
C_{all}	number of possible combinations
c_{corridor}	driving corridor constraint function
c_i	i^{th} constraint
C^n	continuity of n^{th} level
d	path-lateral coordinate
d_i	path-lateral coordinate of the i^{th} trajectory support point
d_{center}	signed normal distance from road centerline
d_{left}	signed normal distance from the left corridor bound
d_{right}	signed normal distance from the right corridor bound
$d_{\text{reference}}$	reference signed normal distance
δ	steering angle at wheels
Δt	trajectory discretization step-width
$\mathbf{g}$	inequality constraints
γ	merit function for step size selection
$\mathbf{h}$	equality constraints
$\mathcal{H}$	Hamiltonian
$j_{\text{acceleration}}$	acceleration summand of the integrand
$\mathcal{J}$	cost functional
$\mathcal{J}_{\text{d}}$	cost function
j_{jerk}	jerk summand of the integrand
j_{offset}	offset summand of the integrand
j_{velocity}	velocity summand of the integrand
κ	curvature
$\kappa_{\max}$	curvature limit of the vehicle
L	integrand of the cost functional
$\boldsymbol{\lambda}$	Lagrange multipliers
l	path-longitudinal coordinate
l_i	path-longitudinal coordinate of the i^{th} trajectory support point
l_{feasible}	feasible range of longitudinal coordinates
l_{safety}	required minimum intervehicular distance
l_{v2v}	intervehicular distance
l_{w}	wheelbase length
m	number of inequality constraints
N	number of timesteps
n	number of optimization parameters

$\boldsymbol{\nu}_k$	Lagrange multipliers at k^{th} step in SQP
o	number of objectives
p	number of equality constraints
ϕ	penalty function for determining step size
ρ_{k_i}	penalty term of i^{th} at k^{th} iteration
$\mathbf{p}_k$	search direction at k^{th} step
ψ	yaw angle
$\mathcal{Q}$	configuration space
$\mathcal{Q}_{\text{free}}$	obstacle-free configuration space
r	radius of the vehicle body
S	feasible solution set
s	signed speed
s_{des}	desired travel speed
s_{max}	speed limit
T	length of planning horizon in seconds
t_i	time at i^{th} timestep
$\mathbf{u}$	control inputs
v	number of vehicles present
$\mathbf{v}_{\text{des}}$	reference velocity for a given road
$\mathcal{W}$	workspace of the robot
$w_{\text{acceleration}}$	weight factor of acceleration
w_{jerk}	weight factor of jerk
w_{offset}	weight factor of offset
w_{velocity}	weight factor of velocity
x	x-component of position in Cartesian coordinates
x_i	x-coordinate of the i^{th} trajectory support point
$\mathbf{x}_i$	trajectory support point of i^{th} timestep
$\mathbf{x}_*$	a local or global optimizer
y	y-component of position in Cartesian coordinates
y_i	y-coordinate of the i^{th} trajectory support point

1. Introduction

This chapter starts with presenting brief history of driver assistant systems and autonomous vehicles with an emphasis on traffic crashes and resulting fatalities. After the impact of driver assistant systems on collision avoidance is clarified, one of the major deficiencies of state-of-art autonomous vehicles is addressed. The stated deficiency constitutes the motivation behind the objective of the thesis. In Section 1.2 the objective is explicitly restated together with the subobjectives that are to be followed for accomplishing the objective. The chapter subsequently introduces contexts of individual chapters and provides an overview to the layout of this work.

1.1. Motivation

Ever since Carl Benz made the patent application of the first automobile on 29 January 1886 in Berlin, the demand and interest on automobiles has increased progressively and automobiles have finally become the primary means of individual mobility. With increasing welfare, the society in any region of the world has demanded more automobiles, where the saturation level on the demand is at about 0.5 car per person on average [1]. Considering the increasing world population and other factors that intensify this trend, automobile sales are expected to increase from 69,1 millions in 2012 to 91,4 millions in 2020 worldwide, which indicates an increase at about 30% [2, p. 5]. The rise in the sales, being an indirect factor of more motor vehicles to be travelling on roadways, leads typically to various environmental hazards, a higher congestion on the available roadways, and even more traffic accidents to occur. A study reveals the actual costs of accidents in the year 2000 to be nearly one trillion dollars [3, p. 2]. A recent study of WHO indicates 1.24 million people around the world have lost their lives in road traffic in 2013, which equals a death rate of 2.35 per minute [4]. Not only the number of traveling vehicles on the roadways, but also the growing older driving population and the fact that driving skills deteriorate with age is an important contributing factor to the fatalities [5].

Driver assistance systems, although first developed with the intention to reduce the duties of the driver mainly for increasing driving-comfort, such as the automatic shifts introduced in 1940s by Oldsmobile, or servo-assisted steering by Chrysler in 1951 etc., have also attained the goal to mitigate roadway-accidents. Statistically, human errors, which mainly result from fatigue, inattention or drowsiness, are with 90% the major rationale of roadway accidents [6, p. 245]. Among most prominent ones of the driver

assistance systems is the Anti-Lock Braking System (ABS) that is first developed by Bosch in 1978. The success of ABS and Traction Control System (TCS) led to the development and become widespread of the Electronic Stability Program ESP, which incorporates the functions of ABS and TCS, besides maintaining stability. In 2004, U.S. National Highway Traffic Safety Administration (NHTSA) has concluded that about one-third of fatal accidents could be prevented by using this technology [7]. In 2009, European Union decided to make ESP mandatory for all new models to be revealed after November, 2011 [8]. Since then here has been a substantial reduction in fatality rates in occured accidents, according to the statistics of NHTSA and of the German Federal Statistical Office [9], [10, p. 440]. This can be explained by the widespread and success of ESP.

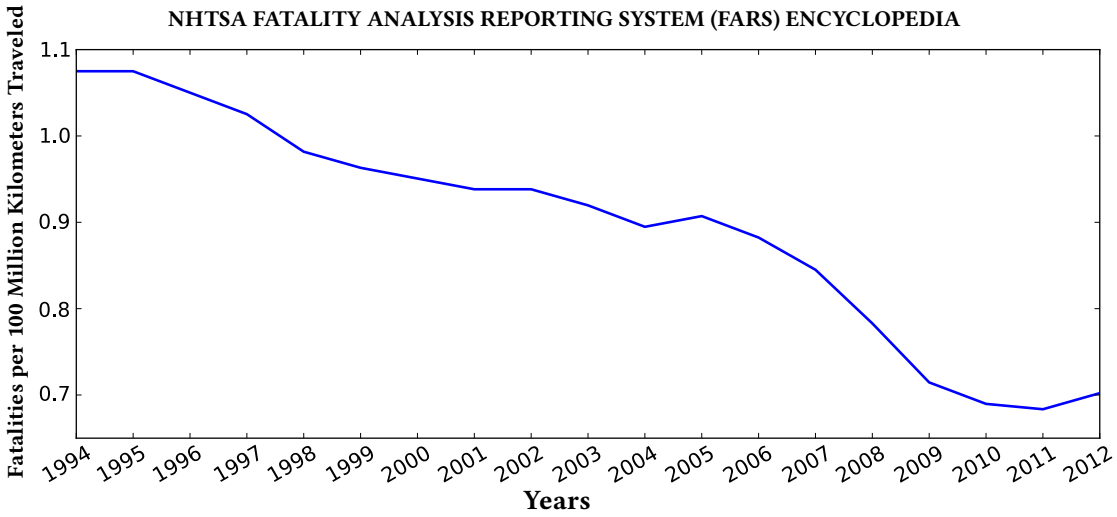


Figure 1.1: Fatality rates per 100 million kilometers traveled among 1994 and 2012 [9].

After the notable benefits of driver assistance systems have become widely known, and have gained market acceptance, the era of next generation assistance systems has begun. Commonly known as Advanced Driver Assistant Systems (ADAS) are defined as "the collection of systems and subsystems on the way to fully autonomous driving" and embrace systems such as Adaptive Cruise Control (ACC), Blind Spot Monitoring, Lane Departure Warning, Lane Change Assistance, Pedestrian Protection Systems etc. These systems, besides increasing safety, provide a higher level of comfort by taking over tertiary, secondary and even primary driver tasks and thereby reduce the strain of the driver [11, p. 11]. Gwehenberger (2010, p.1) claims that if all of the vehicles in Europe would be equipped with ADAS, at least half of the serious accidents could be prevented [6, p. 245].

Mentioned concerns regarding roadway safety and the prestige of being the first innovator and mass producer of those systems has given rise to the research and development investments of automotive companies. According to a research of Bratzel and Tellermann (2011, p.113), the industry spent in 2011 € 48 billion to innovation activities,

amounting an increase of 15% compared to the previous year [11, p. 1]. Germany, with €22 billion research and development expenditures in 2011 and ten accepted patents per day, belongs to the top investors[2, p. 179].

Undoubtedly, the innovation of driver assistance systems has led the industry to focus more on the next technology vehicles and especially to autonomous driving. Efforts towards autonomous driving started in 1961 at Stanford with the 'Cart with Cable' and continued with the introduction of a vehicle of Tsukuba Mechanical Engineering Lab in Japan, which could drive 50 meters on a street by tracking white street markers with speeds up to 30km/h. Contributions from Europe came with the Pan-European PROMETHEUS project, in which Daimler-Benz, University of German Armed Forces and many others participated. The project is renowned for the drive from Munich to Odensee and back, corresponding to a route of 1758 km. During this drive, the vehicle 'VaMP' executed maneuvers to overtake other vehicles and has reached speeds up to 175 km/h, whereas human intervention remained only at 5%. The longest path without human action was 158km, whereas 1 intervention per 9 km was on average. In the same years a similar research has been conducted by team of Carnegie Mellon University in the United States. The project was called 'No hands across America' and could successfully drive from Washington D.C. to San Diego, semi-autonomously; the driver only needed to control longitudinal acceleration [12].

Another trials until the DARPA Grand Challenge were performed from various companies and research labs around the world. Although the first event of the Grand Challenge had no winner in 2004, just a year later, tremendous progress was achieved and many teams performed better than results the best team could obtain in 2004. Short after the Grand Challenge, DARPA announced a new competition, called the 'Urban Challenge' which served as the milestone of autonomous driving. Since then, the issue has become increasingly popular and the lessons learned from those challenges have led to significant advancements in the field. Shortly after, Google introduced its autonomous vehicles which would play a key-role in the establishment of Google Street View. Among other equipments required for cartography, these vehicles have a 360 rotating lidar mounted on the roof for 3D mapping of the environment. As those weird looking vehicles got noticed and become popular among people, and were also proven to be safe enough, legislations to allow those vehicles completely autonomously were issued. The U.S. state of Nevada being the first, followed by states of Florida and California gave license for testing those vehicles on public roadways autonomously.

The development of autonomous vehicles has also involved into the area of race-cars. The autonomous race-car Shelley, developed by Stanford University and Volkswagen of America, inheriting base technology and equipments *Junior3*[1] once had, could successfully complete the Pikes Peak Hill Climb autonomously [13]. The project aimed to preserve the stability of the vehicle at the limits of handling, besides being a precursor of robot-car races that may emerge in the future, the technology could also serve as a foundation for taking over the control of vehicle, whenever the driver cannot maintain

the stability of the vehicle.

Simultaneously with the development of autonomous vehicles, attempts to develop intelligent vehicles that can communicate with each other and with the infrastructure are also made. The California Program on Advanced Technology for the Highway (PATH), led by J. Karl Hedrick, was the first project that studied the issue for a broad scope [14]. In the late 1990s and 2000s several studies for intervehicular communication were performed in the US, Japan and Europe. In 2011, the Grand Cooperative Driving Challenge was the first competition in which heterogeneous vehicles; from trucks to minicars were involved and tests regarding platooning and string stability were performed [15].

Another very substantial advancement has been achieved with *Bertha*, a 2013 Mercedes-Benz S-Class vehicle, for completing a fully autonomous drive on the Bertha Benz Memorial Route in August 2013, 125 years after Bertha-Benz and her two sons did the first cross-country automobile journey in the world. The route covered slightly more than 100 kilometers with a number of highly complex situations, where traffic lights were exposed to direct sunlight and the brightness of red light had become less, various road-intersections and roundabouts, speeds exceeding 50 km/h, a number of traffic participants such as cyclists and pedestrians, and driving corridors that are partially occluded by parking-vehicles. Just 6 years ago, in the Urban Challenge, even though the vehicles were exploiting astonishingly expensive sensors, none of participants had to -and probably could manage to- cope with those real world complications. Unlike the vehicles involved in the Urban Challenge, Bertha was equipped with solely a few additional radar and vision sensors other than its serial-production version beheld. Moreover, those additional sensors were hardly perceivable from exterior, making the vehicle almost indistinguishable from its stock version. The autonomous drive in the tracks of Bertha Benz has been a substantial and promising enhancement to the capabilities of the autonomous-vehicles and redemonstrated the feasibility of autonomous driving [12].

However, even the state of art autonomous vehicle Bertha had some deficiencies. Whenever it came across intersections or roundabouts, where various combinatorial possibilities about sequencing occur, it performed in overall far inferior to the level of an attentive human driver. The sluggish behavior is attributed to still remaining high false-positive rate of the perception system and to the trajectory planning algorithms, which evaluated the combinatorial problems in ad-hoc manner [12]. Combined with the fact that collisions happened in intersections take the major portion with roughly about 31% among all accidents [16, p. 592], existing systems for motion planning at intersection scenarios have to be replaced with methods that employs combinatorial reasoning for planning collision free and optimal trajectories.

[1] The autonomous vehicle of the Stanford Racing Team in the 2007 DARPA Urban Challenge.

1.2. Objective

This work is devoted to inspect combinatorial problems encountered in roadways and aims to propose methods that force the behavior generation system to return optimal reference motion profiles by treating the combinatorial options present.

The objective will be addressed by dividing it into two major subobjectives. The first one is the optimal motion generation in absence of other traffic participants. Although this corresponds to a case devoid of any combinatorial alternatives, it will serve as the foundation for the later study on combinatorial problems. As will be explained in Chapter 2, the sole way of obtaining truly optimal trajectories for a customarily defined optimality criteria is to refer to continuous methods. Once knowledge on optimization based trajectory planning is mature, focus on combinatorial problems and retaining optimality in presence of other traffic participants will be given. This requires integration of combinatorial reasoning into continuous methods and constitutes the second subobjective of the thesis.

The inspection of combinatorial options starts with extracting a representation for all of the combinatorial options available. As simplifying assumptions, the position of all of the obstacles is assumed to be perfectly known, *i.e.* sensor uncertainty is neglected, and overtaking is restricted. Instructions to abandon these assumptions will however be proposed in future work. To find the most favorable combination, all of the combinations will be separately analyzed and optimal motion profiles for the feasible ones will be attained via the use of nonlinear optimization. Hence, finding infeasible combinations, providing an initial guess of the motion profile for the local optimization, and formalization of constraints, which delimit the vehicle motion into a given combinatorial option form the subobjectives of the thesis. Once optimal, real-time applicable trajectories that negotiate safely between the combinatorial gaps in traffic is found, the costs of the combinations will be compared and the most favorable among them will serve as the reference motion for the autonomous vehicle. The derived formulations will be tested and simulated on several non-signalized intersection scenarios created by utilizing an existing geographical mapping framework. Subsequently, the same approach will also be done with the path-velocity decomposition (PVD) and the results will then be compared and evaluated.

1.3. Thesis Overview

The remainder of this thesis is structured as follows.

Chapter 2 elucidates the reason behind utilization of continuous methods for optimal motion planning. First, it recapitulates basic definitions and reviews conventional methods for motion planning. It subsequently focuses on their application on autonomous vehicles and then provides discussions about the degree of optimality they

can yield. Afterwards, it presents nonlinear optimization based motion planning as an alternative and states that ultimate optimality can only be reached by their means. The chapter concludes with presenting a vehicle model for obtaining reference inputs.

Chapter 3 offers a crash course into the theory of mathematical optimization. It starts with the categorization of optimization problems and then continues with introducing the fundamentals of nonlinear optimization and sequential quadratic programming (SQP) based local optimization algorithms. As explained in Chapter 4, the SQP methods are promising solvers for the specific problem present.

Formulation of motion planning as an optimization problem constitutes the topic of the **4th Chapter**. This chapter reveals the objective function and the constraints that maintain a feasible motion. In order to facilitate formalization, the chapter considers an environment free from obstacles. The chapter subsequently presents discretization and initialization required for the utilization of a local optimization algorithm.

Presence of traffic participants and combinatorial problems they bring about are analyzed in **Chapter 5**. This chapter extends the constraints and the initialization method introduced in previous chapter to handle combinatorial options and essentially forms the core of the thesis. Instructions for implementation of path-velocity decomposition is given within this chapter.

The aspects regarding implementation are elaborated in **Chapter 6**. The context of this chapter does not entail providing pseudocodes or any type of application specific aspects. It merely introduces solutions to the problems encountered in implementation and yields information about the geographical mapping framework, and the solver utilized. After the chapter clarifies these issues, it gives an insight into the written simulation program.

Chapter 7 presents and evaluates results of various intersection scenarios simulated. These results comprise figures about optimized motion profiles, and values indicating the optimization process. A comparison of complete and path-velocity decomposed planning, and aspects such as the influence of 'good' and 'bad' initialization and other factors that have an effect on the solution are also discussed in this chapter.

Finally, **Chapter 8** gives a brief summary to the conducted research and obtained results, and provides an outlook for future research.

2. Fundamentals of Motion Planning

This chapter starts with presenting basic definitions in motion planning. Once these are clarified, a review of various motion planning methods is presented. The chapter then focuses on the applicability of the presented methods on autonomous vehicles. Typically, the quality of a planned motion is evaluated according to some predefined criteria. The aspect of quality constitutes the topic of the last section. In this regard, discussions about how well such measures on quality can be satisfied with the introduced methods and how the nonlinear optimization based methods arise as an alternative are presented. The discussion concludes by evaluating the option to integrate vehicle model into the optimization problem and thereby obtaining the reference inputs required for the low level motion controllers directly.

2.1. Basic Definitions

Like any mobile robot, an autonomous vehicle has to perceive its environment and navigate through it in order to complete any given goal. Once perception systems have sensed the obstacles and the structures surrounding the robot, the cognitive system is liable to plan the motion of the robot, under consideration of certain requirements. Finding a motion that fulfills the requirements set is commonly called as *motion planning* and has been addressed as a fundamental problem in robotics.

Based on the requirements set on motion, a distinction is made between path planning and trajectory planning. In *path planning*, which is sometimes referred as the *Piano Mover's Problem*, only the geometric (or the kinematic) constraints are considered. As the most of the mobile robots operate at low speeds and agility is not a fundamental design parameter, planning with only kinematic constraints is generally adequate. However, if this is not case, a more complex planning approach, in which dynamic constraints besides the geometric ones need to be employed. Such a planning method is called as *trajectory planning*, or sometimes interchangeably called as *motion planning*. In this case, path is planned together with speed along the path with C^2 continuity. Motion planning can be, under circumstances, done in an ambient where moving obstacles (commonly referred as *dynamic obstacle* or sometimes as *velocity-obstacle*) are present. If this is the case, for any trajectory to be collision-free, the motion of such obstacles must be taken into consideration, in effect, evolving the motion planning problem to a *kinodynamic planning problem* [17].

Apart from the stated distinctions, there may be some additional requirements that do not necessarily have to be held but, their satisfaction at any time-instance of motion may be beneficial. These type of requirements define the quality or *optimality* of the planned motion. Minimization of execution time, expended energy for cases which respectively *time-optimal*, *energy-optimal* trajectories is of interest, can be given as a trivial example.

Another distinction in motion planning is made between offline planning and online planning. In *offline* planning, the cognitive system (or in other words the *planner*) plans the path before that start of motion. This requires the assumption that all the parameters of the obstacles are perfectly known and the execution of the planned path is exact. Albeit being a plausible assumption for an environment which only comprises static obstacles, if the path of any dynamic obstacle (or sometimes so-called *velocity-obstacle*) may have the possibility to intersect with the path of the autonomous agent, or if the uncertainties and changes in environment of the robot are significant, the assumption loses its viability and in such a case, continuous planning relying on the updates of sensor values during the robot's movement must be conducted. The incrementally generated and updated path planning is called *online* planning.

Especially in the context of autonomous vehicles, planning is divided into two main categories: planning in structured environments and unstructured environments. A very comprehensible definition could be made through the distinction between, streets, which contains directives such as lanes or other type of signs, and places like parking lots, where many such indicators are absent. Environments that exhibit characteristics like the former case is called *structured* environments and the ones that do not are called as *unstructured* environments. It should be further noticed that, within the context of autonomous vehicles, unstructured environments pose a quasi-static problem and the structured environments pose a dynamic problem.

Besides to what extend the optional requirements are met, there are other factors specifying the quality of a given algorithm. Among others, of primary importance are the computational complexity and completeness. With the term *computational complexity*, the dependency of an algorithm's runtime (and sometimes memory requirements) to the "size" of the environment is defined. Generally, algorithms that run in polynomial time at worst case are especially attractive [18, p. 11]. An algorithm is called *complete* if it finds a solution for the motion planning problem whenever a solution exists, and terminates with failure otherwise. However, as the search space of the motion planning problem grows, computation time will rise and completeness may become an arduous task to reach. For such cases, completeness is generally sacrificed and more convenient sub-forms of completeness, such as resolution completeness and probabilistic completeness are defined [19, p. 371].

Apart from the definitions based on algorithms, there are two notions that every motion planning algorithm relies on: the workspace and the configuration space of a robot. The *workspace* $\mathcal{W}$ of a robot is defined as the ambient space of the robot, or in other words: the space in which the robot is located. The *configuration space* $\mathcal{Q}$ (also called C-space)

is the set of all possible configurations that the robot can take. Any configuration of a robot is defined by finite number of parameters used for representing all of its position with respect to an inertial reference [20].

2.2. Overview of Various Motion Planning Methods

In the literature there is a huge variety of approaches for motion planning, each of them having distinct pros and cons. However, in a general, they all share a common trend: austereness on the optimality results with an increasing complexity. Before delimiting the context of the thesis onto autonomous vehicle motion planning, a rough introduction to existing motion planning methods will be given below. Among them, visibility graph method will be referred in the next chapters.

Rapid development in diversity of algorithms can probably be listed as one of the complexities of robot motion planning. Siblings of existing algorithms, providing new features and covering the deficiencies of their predecessors are added up to the literature every day and the borders in between the distinct classes become fuzzier. The classification differs even among the major books of the field. The brief summary will hence be done in parallel with the classification of the book by Howie Choset et. al. [18], but will slight diverge from it, by revising several definitions in the light of recent developments and evaluations from other sources, such as the book by Roland Siegwart *et al.* [19].

2.2.1. Artificial Potential Fields

Artificial potential field methods rely on treating the distance to the goal position as a potential energy function, and thereby, forces the robot to move in a direction that reduces the potential at most. While, the goal position forms the attractive forces, the obstacles in the workspace form the repulsive forces and their coalescence constitute the artificial potential field.

Mathematically denoted, a potential energy function is defined as a continuous function $U : \mathbb{R}^n \to \mathbb{R}$ and the direction of the movement is the direction yielding the steepest descent in the *gradient* of the potential function U. In this case, $\nabla U(q^*) = 0$ corresponds to a critical point, whose type can determined by checking the second derivative, namely the *Hessian* $\mathbf{H}$ matrix of the energy function. At any point, a positive definite $\mathbf{H}$ indicates the point being a local maximum and a negative definite $\mathbf{H}$ indicates a local minimum [18, p. 78].

All gradient based potential field algorithms are plagued by being trapped into a local minimum any other than the global one, whereas the global one corresponds to the goal position. Horseshoe shaped obstacles are the primary source of such problems, [18, p.

89]. However, in the literature there are remedies introduced, of which the most basic one is to follow the negative gradient back whenever such a situation is encountered.

Brushfire algorithm and the wave-front planner are probably the most employed methods among the variants of artificial potential field methods.

2.2.2. Roadmaps

Roadmap theory in motion planning begun with the dissertation of John Canny [21]. Roadmaps are fundamentally alike how humans use highways to drive from a point to an another: in order to reach somewhere, the driver searches for the closest highway and drives to it, completes the bulk motion on the highway until s/he reaches to the point on the highway, that is closest to its goal position. After that point, it gets to its destination by finding a path from the final highway-point. An analogy in robotics is done in such a way that the paths are constructed in advance, and during motion planning the autonomous agent only needs to find a path that has access to an existing path. Thereby, the burden of finding a bulk-path is eliminated and the search problem is reduced. [18, p. 108] Paths created with such an intention are often represented by topological maps, called *roadmaps*, and are gathered by nodes and edges. Nodes can represent any type of remarkable information, whereas edges represent an intersection, or an adjacency, of nodes. Many approaches in motion planning can be grouped under roadmap methods: visibility maps, deformation retracts, silhouettes and even sampling-based methods.

In visibility maps approach, in its simplest form a *visibility graph* is created by using vertex of configuration space obstacles and the start & goal configurations. Then, straight lines between the nodes are drawn and a visibility map is built. Thereby paths yielding shortest possible distance are obtained. Albeit this can be a desirable property returning length-optimal paths, it is more likely to be a flaw, as it causes the robot to get to obstacles as close as possible. A remedy is to add some reasonable safety distance to the obstacles and grow them appropriately. It should be remarked that the application of this approach is limited to configuration spaces with polygonal obstacles. Visibility graph methods are among the fastest planning methods in sparse environments. Increasing number of obstacles, due the increase in number of vertices, will lead to slower runtimes.

Deformation retracts tend to find a line that is equidistant to the obstacles in the ambient. The most renowned method is the *Voronoi diagram*. In silhouette approach, shapes or *silhouettes* of objects are defined as a one-dimensional function and the free-workspace is sliced at the extrema of it. Such an extremum is called *critical point* and roadmaps are generated by using them.

2.2.3. Cell Decompostions

Path planning in cell decompositions start with discretizing, or *decomposing* obstacle-free-workspace into regions called *cells*. Once the decomposition is completed, path planning starts with determining the cells in which the start and the goal points lie. Afterwards, according the adjacency of cells, some graph search algorithms such as breadth-first, depth-first, Djikstra etc. are run and thereby the path planning is done.

Cell decomposition methods are generally used for *coverage*, where it is important for the agent to move over all of the points in a free space. Trapezoidal decomposition, Morse decompositions, visibility-based decompositions for pursuit/evasion, and the state lattice graphs are the most significant variants of cell decomposition methods.

2.2.4. Sampling-based Methods

Sampling-based methods are first introduced S.M. LaValle in the late 1990s, as an efficient alternative for roadmap methods. In high dimensional path planning problems, the complexity of roadmap generation increases and hence, it becomes intractable to find a solution within a limited amount of time. In such cases, it is favorable to create a roadmap by searching the configuration space randomly and checking whether a single robot configuration is in $\mathcal{Q}_{\text{free}}$ or not. Thereby, without explicitly constructing the boundaries of the configuration space obstacles, but by merely accessing the configuration space obstacles, solutions could be obtained very fast [18, p. 201].

A very attractive property of randomized search techniques is that, the motion planning can still be performed under kinodynamic constraints, and as a result, can successfully be applied to variously constrained systems. There have been tremendous research in sampling based-methods and now there are dozens of randomized search methods available. Most of them posses the desirable property of being *probabilistically complete*, indicating that as the number of searches goes towards infinity, the algorithm will eventually find a solution.

Among its variants, Probabilistic RoadMap Planners (PRM) are widely used for multiple-query planning, whereas for many planning applications the answer for a single query is important, *i.e.* whether the given configuration is feasible or not. This is often the case in kinodynamic planning, where the initial and the goal configurations are known and quests about intermediary configurations are to be performed. Expansive-Space Tree planner (EST) and Rapidly-exploring Random Tree planner (RRT) and their descendant variants are suitable for such queries. For very difficult path planning problems, their combination, the Sampling-Based Roadmap of Trees (SRT) is used.

The other prominent alternative sampling-based motion planning method is the recently adopted lattice search. This approach is based on selecting state lattice samples from a precomputed graph of feasible maneuvers that is created for a discretized configuration space of the robot. Due to the discretization, they are sometimes grouped

under cell decomposition methods. The precomputed alternative samples are stored in memory and during execution, the best one is selected with the aid of graph search algorithms. The distinguishing property of lattice search from other cell decompositions is that, the constraints influencing the path of the robot is intrinsically implemented in the lattice generation phase.

2.3. Application of Various Motion Planning Methods on Autonomous Vehicles

Autonomous vehicles diverge from other forms of autonomous agents with the constraints they have to obey. First and probably the most fundamental constraint arises from the form of being a "car-like vehicle". The restriction on sideway translation limits the values that the velocity can take and, in end effect, prevents the velocity values to be integrated into configuration constraints. Such constraints are referred to as *nonholonomic* constraint.

There are also other factors delimiting the motion of the vehicle, like the Ackermann angle (or the minimum turning radius), steering rate, tire friction limits and the acceleration limit arising out of it. Moreover, the demands on a comfortable ride and the high dimensional, kinodynamic obstacle containing structure of the environment impose additional constraints which are to be taken into consideration. Not the vast majority of the motion planning algorithms, however, are able to observe such requirements during planning and hence, only several types can come to the fore.

Among them, RRT is the most widespread variant. Papers reporting its applications can be found in [22], [23], [24] and in many others. It should however be noted that all of the randomized sampling algorithms necessitate post-processing in order return applicable trajectories. Systems motivated from lattice search methods, which recently have become popular, can be found in [25], [26], [27], [28] and many others. Apart from the papers published, the book edited by J.P. Laumond is entirely devoted to trajectory planning for nonholonomic systems [29], [18, p. 442].

2.4. Optimal Trajectory Planning

The progress in engineering have always brought about the effort to optimize existing systems and products according to certain criteria. This also applies to trajectory planning, in which optimality in terms of some predefined criteria is a key effort to reach. As aforementioned, there are different evaluations of optimality: given initial and goal positions, an algorithm may be optimal in sense of generating the shortest feasible path, or returning a path that takes the least time to reach the goal, *i.e.* a time-optimal trajectory. Although many existing motion planners in field robotics define their path

quality based on such strict definitions, in reality, such definitions make rarely sense and an optimality criteria comprised of various aspects is more likely to be of interest. For example, besides time optimality, consumed energy could be a secondary design criteria and in case, could lead to trade-off in time-optimality to a certain extend. Or as a further example from our context, the ride comfort is an essential element which needs to be integrated into the optimality measure. The sole way of obtaining a path, with regard to any customarily defined criteria is to employ cost functionals and perform optimization, either as a reshaping agent in post-processing of a path obtained from the previously indicated methods, or as a stand-alone approach.

Before focusing more throughly on trajectories based on custom optimality definitions, common optimal path planning methods for autonomous vehicles will be briefly reviewed in terms of optimality. A traditional approach is to treat a vehicle as either a Dubins' or Reeds-Shepp's car. A limiting factor of those approaches was that they were defined for obstacle-free environments, but some papers extended these to dynamic environments [30, p. 729], [31], [32]. However, although they become utilizable in dynamic environments after these extensions, they are only optimal in sense of delivering the shortest path possible, *i.e.* not a custom objective function or even time-optimality.

Sampling based methods, which have dominated the past decade of motion planning, are based on sampling the configuration space either randomly or deterministically. After the samples are generated, best trajectory with respect to any given criteria is selected. The randomized methods, such as the RRT, return a trajectory sampled by happenstance and hence, even after postprocessing, any solution obtained could be far away from the true optimum. Although RRT algorithms seem to be a promising way for motion planning, they have significant drawbacks of having non-deterministic execution times and being unexceptionally suboptimal, which is a result of sacrificing complete search for faster execution times.

The lattice search, which is the sole representative of deterministic sampling, in the contrary, is not plagued by randomness. But like any other cell decomposition method, this approach also requires a certain degree of discretization of the workspace, leading the accuracy, and accordingly the optimality, of the path being restricted to the sampling resolution. Although reducing the grid size appears as a remedy and as a way to obtain near optimal trajectories, this will yield tremendous search space and eventually results in excessive search time [33], [34].

An another way to find trajectories that are optimal in terms of a customarily defined objective function is to refer to methods that are based on employing *nonlinear programming* (NLP) (or *nonlinear optimization*). These methods, which sometimes are called as *trajectory optimization*, do not discreticize the workspace, but rather perform continuous optimization to find the trajectory parameters for any given time. Thereby, they do not introduce any inaccuracy that would intrinsically arise due to discretization. The continuous search space brings about the opportunity to obtain *truly* optimal solutions. Optimization of the objective function (or the *cost functional*, *i.e.* integrals involving

functions and their derivatives) is performed for timesteps generated by parameterizing time into equal step-lengths, under restriction of applied constraints.

Transforming trajectory planning problem into such an optimization problem usually returns a nonlinear quadratic program, which can be best solved through the utilization of sequential quadratic programming (SQP) methods. Such theoretical aspects of these methods will be explained in Chapter 3, but for short, they utilize gradient descent to determine the search direction that reduces the value of the cost function at most. Through that, a cost function defined in accordance with general pattern of vehicle-motion will result in desirable reference trajectories.

A property of these methods is that, they share the same solution methods with optimal control problems. Optimal control inputs of any system are usually found by transforming optimal control problems into parameter optimization problems [35]. A conceptional overview of the application of such a method functions can be given as follows. The functional that is to be optimized can be given as:

$$\mathcal{J} = \int_0^{t_f} \mathcal{L}(\mathbf{x}(t), \mathbf{u}(t), t)\, \mathrm{d}\mathbf{x}\,, \tag{2.1}$$

where the integrand $\mathcal{L}$ is representing the *Lagrangian*, which is a generalization of the Lagrangian used in dynamics. Afterwards, to be able to include the constraints set on motion, the *Hamiltonian* $\mathcal{H}$ is defined. Integrating the constraints can be done using Lagrange multipliers, just like it is done in the case of nonlinear optimization, which will be explained in next chapter. However, in optimal control lingo, although there is not any practical difference, these multipliers are defined as *costate variables*. The set of optimal control inputs can be found by numerically solving the values of those multipliers. According to *Pontryagin's minimum principle*, any optimal solution must minimize the value of the $\mathcal{H}$. This actually represents the first order necessary condition:

$$\frac{\partial \mathcal{H}}{\partial \mathbf{u}} = 0\,. \tag{2.2}$$

The sufficient condition is obtained from the convexity condition, or the *Legendre-Clebsch* condition;

$$\frac{\partial^2 \mathcal{H}}{\partial \mathbf{u}^2} > 0\,, \tag{2.3}$$

whose left hand side is the Hessian of the Hamiltonian [18, p. 386], [30, p. 708], [36]. This formulation of nonlinear-optimal control has in fact the same structure as an arbitrary constrained non-linear optimization problem and therefore evokes questioning the incorporation of low-level control actions to upper level, or the *behavioral layer*. In such a case, the parameters of the optimization problem would also be the action variables of the system and thereby, the requirement on implementing an additional feedback tracking controller would be saved. Whenever the vehicle would deviate from its reference path, presumably as a result of any external excitation, the optimal-controller would adjust the control input in such a way that it would regulate the vehicle back to

its optimal path. However, it was shown that decoupling trajectory tracking from planning, and dealing with them in separate layers is advantageous [25, p. 6]. Thus, tracking of a path generated in the behavioral layer can be done with a separate longitudinal controller. The lateral offset can then be eliminated through the use of a lateral tracking controller that is based on vehicle kinematic model.

For modeling the kinematics of a vehicle, bicycle model is usually sufficient. However, the basic bicycle model allows instantaneous changes of steering angle. Although this may be applicable to a motion at very low speeds, such as for parking maneuvers where quasi-static motion is present, at high speeds it loses its validity. In such cases, improvement on the model can be obtained through introducing an integrator to the action variable, namely to the steering angle and hence, continuous change in the steering angle can be secured. For such a model, the state of the vehicle can be represented by (x, y, ψ, s) and control inputs can be defined as $(\dot{s}, \delta)$, where x and y denote the vehicle position in Cartesian coordinates, ψ the vehicle orientation or *yaw* angle, s the longitudinal velocity, or the *signed speed*, $\dot{s}$ the longitudinal acceleration at the center of the rear axle, and δ the steering angle at wheels. This model is sometimes referred as *continuous-steering car* [30, p. 614]. Notice that, high lateral accelerations will cause considerable slip angles at tires to occur, and will eventually fail to yield accurate results.

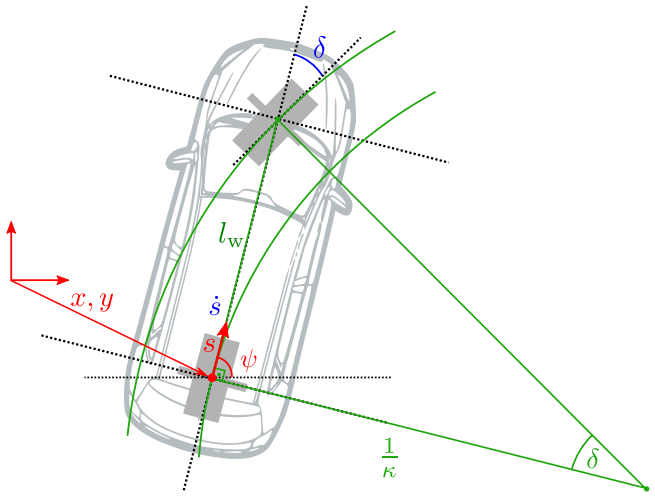


Figure 2.1: Kinematic one track model of a vehicle [37].

Some of the dynamic systems, enclosing car-like vehicles, have the property called *flatness*, or sometimes referred to as *differential flatness* [38], [18, p. 447]. For such systems, all states and inputs can be expressed with a set of *flat* outputs, which is in the same dimension of the inputs, and their time derivatives [39]. This yields

$$x, y, \psi, s, \delta, \dot{s} \longleftrightarrow x, y, \dot{x}, \dot{y}, \ddot{x}, \ddot{y}\,, \tag{2.4}$$

where the ψ, s, δ, $\dot{s}$ can be obtained from the kinematic vehicle model, as shown in

Figure 2.1

$$\psi = arctan\left(\frac{\dot{y}}{\dot{x}}\right) , \tag{2.5a}$$

$$s = \left\|(\dot{x}, \dot{y})^{\mathrm{T}}\right\| , \tag{2.5b}$$

$$\delta = arctan(l_{\mathrm{w}} \cdot \kappa) , \tag{2.5c}$$

$$\dot{s} = \left\|(\ddot{x}, \ddot{y})^{\mathrm{T}}\right\| . \tag{2.5d}$$

l_{w} and κ denote vehicle wheelbase length and curvature respectively. Thereby, once a trajectory that minimizes a cost-functional is found, the state variables and the tracking control inputs can be easily derived using the equations given above. Together with the state variables, these will serve as input to the lateral and longitudinal motion controller.

3. Theory of Optimization

A recapitulation of existing motion planning methods were given in the Chapter 2. In that chapter, it was clarified that a trajectory that is truly optimal in sense of minimizing any customarily defined goal function is only achievable by the utilization of continuous methods. Before inspecting how the trajectory planning problem can be converted into an extremum problem, it is beneficial to review the essentials of mathematical optimization.

The review starts with introducing basic definitions and the categorization of optimization problems. Afterwards, it continues with focusing on the theory behind the algorithms used for solving the special type of optimization problem undertaken. The survey, however, is restricted to the theoretical backgrounds and hence, is presented without addressing properties of the application specific problem. Any information about the structure of the problem, will be presented in Chapter 4.

3.1. Basic Definitions in Optimization

A mathematical optimization problem has the form,

$$\min_{\mathbf{x}\in S} f(\mathbf{x})\,, \tag{3.1a}$$

where the $f : \mathbb{R}^n \to \mathbb{R}$ is the *objective function*. The vector $\mathbf{x} \in \mathbb{R}^n$ represents the n optimization parameters and define the dimensionality of the problem. In some circumstances, the values that the parameters can take may be limited. Such a limitation on the parameter values is called *bound constraints* and is defined as

$$\mathbf{lb}_i \leq \mathbf{x}_i \leq \mathbf{ub}_i\,. \tag{3.1b}$$

The $\mathbf{lb}_i$ and $\mathbf{ub}_i$ represent the lower bounds and upper bounds respectively. An optimization problem may additionally be subject to a set of constraints of the form:

$$g_i(\mathbf{x}) \leq 0,\ \ i = 1, \ldots, m \tag{3.1c}$$

$$h_i(\mathbf{x}) = 0,\ \ i = 1, \ldots, p\,. \tag{3.1d}$$

The g_i represents the inequality constraints, and h_i represents the equality constraints. If a point satisfies all of the constraints employed on the problem, the point is stated

as a *feasible point* and the set of all feasible points is stated as the *feasible region* or the *feasible set*. A feasible set will be denoted by S. It should be noticed that inequality constraints reduce the size and the equality constraints reduce both the size and the dimensionality of a feasible region.

The equations specifying the objective function and the constraint functions, can be *linear* or *nonlinear*. If the output of any of the constraint functions or the objective function is not proportional to the input, that is, if the superposition principle

$$f(\mathbf{x}_1 + \mathbf{x}_2) = f(\mathbf{x}_1) + f(\mathbf{x}_2)\,, \tag{3.2a}$$

and the homogeneity

$$f(\alpha \mathbf{x}_1) = \alpha f(\mathbf{x}_1) \tag{3.2b}$$

does not hold, the problem at hand is a *nonlinear optimization* problem. Most of the systems in science and engineering arise as nonlinear problems.

Convexity of a function plays an important role in the optimization theory. If a line segment connecting any two points in a set $C \subset \mathbb{R}^n$ lies also in C, *i.e.* if any $\mathbf{x}_1, \mathbf{x}_2 \in C$ and any $\boldsymbol{\theta}$ with $\mathbf{0} \leq \boldsymbol{\theta} \leq \mathbf{1}$ satisfies

$$\boldsymbol{\theta}\,\mathbf{x}_1 + (\mathbf{1} - \boldsymbol{\theta})\mathbf{x}_2 \in C \tag{3.3a}$$

then the set C is *convex*. Like the definition of a convex set, the convexity of a function can be defined. A continuous function $f(\mathbf{x}), \mathbf{x} \in C$ is *convex* if $\forall\ \mathbf{x}_1, \mathbf{x}_2 \in C$, and $\mathbf{0} \leq \boldsymbol{\theta} \leq \mathbf{1}$ the inequality

$$f(\boldsymbol{\theta}\mathbf{x}_1 + (\mathbf{1} - \boldsymbol{\theta})\mathbf{x}_2) \leq \boldsymbol{\theta} f(\mathbf{x}_1) + (\mathbf{1} - \boldsymbol{\theta}) f(\mathbf{x}_2) \tag{3.3b}$$

holds. Convex objective and constraint functions constitute a convex optimization problem. More information about convex functions and their optimization can be found in [40, p. 35], including discussions about which of the mathematical operations preserve convexity.

Regarding whether the gradient of a function is defined globally and is continuous, a distinction is made between *smooth* and *non-smooth functions*. Class of continuity is often denoted by C^n, where n represents the class of continuity. Smooth functions, due to their recursive differentiability, are sometimes denoted with C^∞ – class continuity. The issue is depicted in Figure 3.1: the function f_1 is a smooth function, and the f_2 is a C^0 class continuous function, hence is non-smooth. However, smoothness practically is attributed to any function that fulfills C^2 continuity, which implies continuity of the functions first and second derivatives [41, p. 6].

Figure 3.1: Illustration of a smooth and non-smooth function [42]. It should be remarked that both of the functions are continuous.

A solution of the optimization problem, which commonly is denoted by $\mathbf{x}_*$, can be either a *global* optimizer or a *local* optimizer. A global optimizer minimizes the objective over all feasible points, whereas a local optimizer minimizes over the feasible points in the vicinity ($\boldsymbol{\epsilon}$) of $\mathbf{x}_*$. Their mathematical expression can be given as

$$f(\mathbf{x}_*) \leq f(\mathbf{x}) \; \forall \mathbf{x} \in S \tag{3.4a}$$

for global optimizers, and for local optimizers

$$f(\mathbf{x}_*) \leq f(\mathbf{x}) \; \forall \mathbf{x} \in S, \text{ such that } \|\mathbf{x} - \mathbf{x}_*\| < \boldsymbol{\epsilon}\,. \tag{3.4b}$$

Figure 3.2 shows the minima: x_b and x_c correspond to local minimizers and x_a corresponds to the global minimizer.

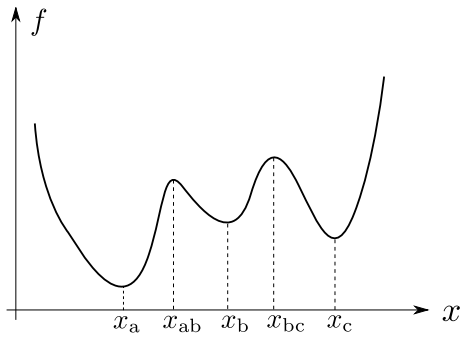


Figure 3.2: Illustration of global and local minimizers for a univariate function.

Whether the objective is to find a local or a global minimizer, optimization algorithms are separated into two classes: *global optimization* and *local optimization*. Global optimization algorithms trade off finding the global optimum to run time. The difficulty of finding the optimum parameters grow exponentially as the number of optimization parameters n increase [40, p. 10]. Because local optimization algorithms have better runtimes than the global optimization algorithms and since they can solve problems of larger scale more efficiently, they are especially for nonlinear optimization problems widely applied.

A significant drawback of local optimization algorithms is their requirement on a reasonable initial guess to be made. Absolutely the initial guess determines to which local

minima the algorithm will converge. Therefore, the initial guess and the constraints, by confining the parameters, play a key role on the quality of the solution.

Local optimization algorithms are especially attractive for convex problems. An optimization problem, whose objective and the inequality constraint functions are convex, will have a single local minimum, and in this case, any obtained solution will correspond to that minimum, which is also the global one [41, p. 6].

Apart from the global/local distinction, local algorithms are separated into two classes depending on whether they use gradient information or not. If the problem at hand, *i.e.* the objective and constraint functions, is differentiable and their gradients are easy to determine, then it is beneficial to use that gradient information during optimization. The gradient is the partial derivative of the function with respect to each parameter, and describes how the function value reacts to a change in the parameter values. It facilitates finding out the descent and ascent directions of the objective function, whose incorporation into the optimization algorithms leads to significantly lower computation times. Such algorithms are called *gradient-based algorithms*.

Some problems, however, may be hard to differentiate, which in result making the analytic gradient calculation to become very hard. In such cases there are two remedies: either to employ finite-difference methods, or to use automatic differentiation tools. Finite difference methods evaluate the derivative of a function for a given input using the Taylor's series. Such a derivative calculation has two significant problems. First, the derivative approximation is dependent on the input given. That means, the whole calculation must be redone, if derivative at a different point is sought for. Second, as a result of being based on Taylor's series, they are subject to truncation and round-off errors, and can therefore yield only an approximation of the true derivative. It should be noticed that, the differentiation operation will amplify any non-smoothness in the problem. This has the potential to plague any Hessian approximation derived from numerically calculated derivative. In circumstances, the amplified errors in the Hessian may even cause the algorithms to fail [41, p. 422].

The alternative option, automatic differentiation methods apply a different approach. They evaluate a given function and, commonly using chain rule, compute an equivalent function together with its analytical derivative. Once the analytical derivative is obtained, they return more accurate answers with lower computational costs. The drawback of these methods is the requirement to store all of the intermediate results during the evaluation of the optimization problem [41, p. 429].

In cases, where the gradient is hard to obtain or is not reliable, algorithms that do not rely on the gradient information, namely *derivative-free algorithms*, are recommended to be utilized. Nevertheless, the unavailability of the gradient information leads to considerably longer computation times. The most renowned derivative-free algorithm is the *Nelder and Mead* (or the *Simplex*) method.

Objective functions modeling physical systems or real life incidents can rarely be expressed with a single function. Usually there are conflicting objectives. A value de-

creasing the cost of a single objective simultaneously exacerbates the cost of other objectives. Thus, the optimality of any solution can be defined in sense of the trade-offs done among these conflicting goals. Such an optimization problem is called *multiobjective optimization*, *multicriteria optimization*, or sometimes *Pareto optimization* and any optimal solution obtained is called a *Pareto-optimal* solution.

There are three types of solution methodologies. Probably the most commonly used one is the weighted sum approach, which is classified under the a-priori methods [43, p. 7]. In this approach, all of the objective functions ares multiplied with a *weight* representing their relative significance. Then, an objective function is gathered from the sum of weighted objective functions. The new objective function can eventually be optimized using existing optimization methods.

For a problem having o objectives, the multiobjective function can be given as

$$\min_{\mathbf{x}\in S} \sum_{i=1}^{o} \alpha_i f_i(\mathbf{x}) \,, \tag{3.5a}$$

where $\alpha_i = 1, \ldots, o$ represents the relative weight of each objective and satisfies

$$\boldsymbol{\alpha} \geq \mathbf{0} \mid \sum_{i=1}^{o} \alpha_i = 1 \tag{3.5b}$$

[44, p. 160].

Like in other forms of numerical methods, any solution found will only be an approximation of the real value. How close a solution should be to the real value determines the termination point of an algorithm. However, solution is the element that is sought for and hence is unknown at the beginning. Therefore, not the solution itself but, how much the value of a solution candidate changes from one step to another can alternatively be exploited. Depending on whether the change, with other words defined as the *tolerance*, in the optimization parameters or in the function value is observed, it is distinguished between *parameter* tolerance and *function value* tolerance. A further classification on tolerance condition is the *absolute* and *relative* tolerance. They are calculated as:

$$\text{Absolute tolerance} = |\Delta \mathbf{x}_i| = |\mathbf{x}_i - \mathbf{x}_{i-1}| \,, \tag{3.6a}$$

$$\text{Relative tolerance} = \left| \frac{\Delta \mathbf{x}_i}{\mathbf{x}_i} \right| = \left| \frac{\mathbf{x}_i - \mathbf{x}_{i-1}}{\mathbf{x}_i} \right| \,. \tag{3.6b}$$

Apart from the tolerance conditions stated above, the elapsed computation time or the number of function evaluations can be used as a termination criteria as well.

Within the context of this thesis, as explained in Chapter 4, a nonlinear, convex function is solved using gradient-based local optimization algorithms. Therefore, the work continues with presenting appropriate algorithms.

3.2. Nonlinear Optimization

Like most of the problems in engineering, the trajectory optimization problem is nonlinear. The theoretical analysis of nonlinear optimization methods start with considering an optimization problem without any constraints. This facilitates understanding the fundamentals of a very general case, which eventually serves as a foundation for constrained optimization algorithms. Once an insight into the unconstrained case is gained, equality constrained problems are transformed into a form such that they can be solved using the same fundamental procedure. A similar approach is applied for inequality constrained problems. This section will only introduce the fundamentals, without delivering their derivations. Necessary derivations and proofs can be found in almost every textbook, such as in [45], [41], [44].

3.2.1. Unconstrained Optimization

In the Section 3.1 the fact that the local optimization algorithms are only able to find a minimum in the vicinity (ϵ) of a given point $\mathbf{x}_*$ was mentioned. In gradient-based methods, for function f of $\mathbf{C}^2$ continuity, whether any point in the vicinity yields a greater minimum can be checked efficiently by using the gradient $\nabla f(\mathbf{x}_*)$ and the Hessian $\nabla^2 f(\mathbf{x}_*)$. For such a function, the second-order Taylor series of a point $\mathbf{x}_*$ can be formulated in remainder form as

$$f(\mathbf{x}_* + \mathbf{p}) = f(\mathbf{x}_*) + \nabla f(\mathbf{x}_*)^{\mathrm{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathrm{T}}\nabla^2 f(\boldsymbol{\xi})\mathbf{p}\,, \tag{3.7}$$

where $\mathbf{p}$ represents the feasible search directions, which in an unconstrained case can take any value. Skipping the derivations, it can be shown that, for a point $\mathbf{x}_*$ to be a minimum, it has to satisfy

1. $\nabla f(\mathbf{x}_*) = \mathbf{0}$, *i.e.* $\mathbf{x}_*$ be a stationary point
2. $\det[\nabla^2 f(\mathbf{x}_*) - \boldsymbol{\lambda}\mathbf{I}] = 0,\ \forall \lambda_i \geq 0$, *i.e.* $\nabla^2 f(\mathbf{x}_*)$ positive semidefinite.

These conditions are called *first-order necessary condition* and *second-order sufficient condition*, respectively. However, there are cases in which sufficient conditions are not necessary, as in the case of $f(x) = x^4$, where the Hessian matrix vanishes [45, p. 16], [41, p. 360].

The *Newton-Raphson method* (or just *Newton's method*) is a method for finding iteratively better approximations of a value, to find the roots of a function. Once an estimate of the solution x_k of the function f is given, by using the first two terms of the Taylor series of f, a linear approximation by drawing geometrically a tangent line from the function value at x_k is done. The point, where the tangent line crosses x-axis is taken as the new value x_{k+1}. This process is repeated iteratively until the change in the values falls below the predefined tolerance criteria. Like for any nonlinear function, the Newton's method can employed to first-order necessary condition to the value of x_*. The

utilization of Newton's method for optimization is sometimes called *Newton's method in optimization*. Within concern of this work, it will simply be referred as Newton's method.

The Newton's method for multidimensional case is commonly formulated as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k)\,. \tag{3.8a}$$

A more succint formulation can be given as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k\,, \tag{3.8b}$$

where $\mathbf{p}_k$ is the solution to the *Newton's equations* and represents the search direction. Instead of the form given in (3.8a), the computation of $\mathbf{p}_k$ is done by solving linear equations in place of computing the inverse of $\nabla^2 f(\mathbf{x}_k)$:

$$[\nabla^2 f(\mathbf{x}_k)]\mathbf{p} = -\nabla f(\mathbf{x}_k) \tag{3.8c}$$

[41, p. 365].

Newton's method has a very desirable property. If the initial estimation is close enough to solution, it converges quadratically. The method unfortunately has two very crucial drawbacks that overshadow the aforementioned property. If the initial estimation is far away from the solution sought for, it will likely converge very slow. In some circumstances it may even not converge at all. A further case to be considered are the costs related to use of Newton's method. In the preceding form, or so-called in its *classical form*, Newton's method requires the Hessian $\nabla^2 f(\mathbf{x})$, which can be an expensive process to calculate. It requires $O(n^2)$ derivative computations for obtaining the Hessian, $O(n^2)$ storage locations for storing it, and $O(n^3)$ arithmetic operations for solving the Newton formula. So, once the Hessian is derived, the computational cost of solving the linear system is $O(n^3)$ times the cost of evaluating f. As these are the costs per iteration, for a problem of moderate size ($n < 200$) storing the matrix can be acceptable, whereas the computational cost might probably be not [41, p. 401]. Therefore, alternative methods motivated from Newton's method are developed.

Cost related problems that arise due to computing and storing the Hessian can be resolved by using some approximation instead of Hessian. For such a case, the search direction can be determined by using the approximation $\mathbf{B}_k$, which is a positive-definite matrix:

$$\mathbf{B}_k\,\mathbf{p} = -\nabla f(\mathbf{x}_k)\,. \tag{3.9}$$

It should be noted that, this equation has the same form as the Formula (3.8c). The difference is the approximated $\mathbf{B}_k$, which is in case of Newton's method equal to $\nabla^2 f(\mathbf{x}_k)$. For very large problems, not only the computation complexity, but also the huge storage requirements may necessitate to skip over the use of Hessian by taking $\mathbf{B}_k$ as an identity matrix. Such an approximation is called *Steepest-Descent* method. In fact, it has a

quite different derivation than it is stated above, but for simplicity, it can be considered as an identity matrix.

Although approximating the Hessian may seem beneficial and logical at first, if its task is to be considered, it would be evident that the cons overweight the pros. The primary task of the Hessian is to provide a better estimation of the search direction by modeling the change in the gradient of the function. As a result, any approximation done on it will compromise on the main benefit it brings about. To state it more explicitly, any approximation on Hessian will sacrifice the quadratic convergence to a certain extend.

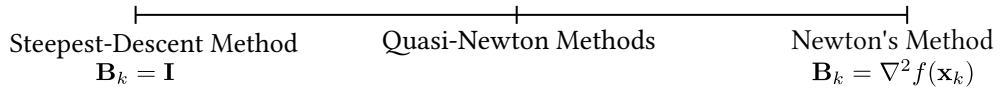


Figure 3.3: Hessian approximation methods.

The impracticality of both methods have led to the development of "hybrid" approximation methods, the so called *quasi-Newton* methods. The family of quasi-Newton methods beholds various methods with disparate approximation formulas. Nevertheless, they are all based on the finite difference approximation of the secant formula. Skipping the derivation, a very popular quasi-Newton algorithm is given by the formula

$$\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{(\mathbf{B}_k \mathbf{s}_k)(\mathbf{B}_k \mathbf{s}_k)^{\mathrm{T}}}{\mathbf{s}_k^{\mathrm{T}} \mathbf{B}_k \mathbf{s}_k} + \frac{\mathbf{q}_k \mathbf{q}_k^{\mathrm{T}}}{\mathbf{q}_k^{\mathrm{T}} \mathbf{s}_k}, \tag{3.10}$$

where $\mathbf{s}_k = \boldsymbol{\alpha}_k \, \mathbf{p}_k$ and $\mathbf{q}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$. The algorithm is named after its discoverers Broyden, Fletcher, Goldfarb and Shanno, and is briefly called the *BFGS algorithm*, Iteratively approximating $\mathbf{B}_{k+1}$ from $\mathbf{B}_k$ in the consideration of the additional knowledge gained from that step allows the application of Cholesky factorization. This reduces $O(n^3)$ arithmetic operations required for solving the linear system of equations at each step to $O(n^2)$. The gain obtained in the computational complexity is not in a huge expense of convergence, like in steepest-descent; BFGS converges superlinearly. Superlinear convergence is defined as $1 < r < 2$ for

$$\lim_{k \to \infty} \frac{\|\mathbf{e}_{k+1}\|^r}{\|\mathbf{e}_k\|} = C < \infty, \tag{3.11}$$

where $\mathbf{e}_k$ corresponds to $\mathbf{x}_k - \mathbf{x}_*$. In the case of quadratic convergence $r = 2$ [41, p. 59].

With the BFGS-update algorithm, the properties of Hessian, such as the symmetricalness or the positive definiteness, can also be imposed on the approximation $\mathbf{B}_k$. This makes the quasi-Newton methods to become especially suitable for quadratic optimization problems [41, pp. 414, 417]. So will one of the two major problems of the Newton's method be rectified. An illustration of steepest descent method and the Hessian illustration is given in Figure 3.3.

There is a very important difference between Newton's method and steepest descent method for several type of problems. If the solutions of the parameters of a multiparameter optimization problems are too separate from each other, *i.e.* for example, while, $x_1 \approx 10^{-8}$, $x_2 \approx 1$ and $x_3 \approx 10^{8}$, then the problem is indicated as *poorly scaled.* In such a case, equal amount of changes in parameters will return higher differences in the value of the objective function. It is not hard to imagine that, such a case corresponds to elliptical contour lines of the objective function. As the steepest descent method uses only the gradient information while finding the direction, it geometrically takes a step that is normal to the contour line at that point. In case of elliptical contours, at the sides where the decrease in the function value is low, this causes "zigzagging", and consequently, the steepest descent method to converges very slowly. On the other hand, the Newton's method, by virtue of the incorporated Hessian, is unaffected by it [45, p. 27]. Therefore, if steepest descent or quasi-Newton methods would be used in the solution process, and the solution of the parameters are expected to differ substantially, prescaling must be done. However, some of the quasi-Newton methods do the scaling by itself [46].

For guaranteeing convergence, there are two main strategies: the *line search* and *trust region.* These are sometimes referred as *globalization strategy* and provide a safety net, against the case the algorithms diverge. Both essentially try to follow the fundamental principle of decreasing the function value at each iteration. Its mathematical expression can be given as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \boldsymbol{\alpha}_k \, \mathbf{p}_k \,, \tag{3.12a}$$

so that

$$f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k) \,. \tag{3.12b}$$

In the line search methods, or *linear search* methods, the algorithm uses the Newton direction to search for a new point $\mathbf{x}_{k+1}$ by creating a line $\mathbf{y}(\boldsymbol{\alpha}) = \mathbf{x}_k + \boldsymbol{\alpha}_k$, and chooses an appropriate step-width $\boldsymbol{\alpha}_k$, such that it causes the function value to decrease. For the special case of Newton's method, $\boldsymbol{\alpha}_k$ is equal to $\mathbf{1}$. Trust region methods, on the other hand, follow a quite different strategy. They use the information gained about f at the current point $\mathbf{x}_k$ to construct a model and then choose the maximum step-width $\boldsymbol{\alpha}_k$ – or more specifically the trust-region radius Δ_k – and select the direction of the step $\mathbf{p}_k$ that reduces the function value at most. If the obtained results are unsatisfactory, the process is repeated by selecting a smaller Δ_k.

Although the performance of these methods is dependent on the individual problem, their performance on average is equal [41, p. 394]. The algorithm used in the thesis uses a method that can be categorized as line search method. Many of such methods make four assumptions in total, two on search direction $\mathbf{p}_k$ and two on the step length $\boldsymbol{\alpha}_k$ to guarantee convergence. These are: $\mathbf{p}_k$ produces "sufficient descent" and is "gradient related", and $\boldsymbol{\alpha}_k$ produces "sufficient decrease" and is not too small. These lead to the

definition of *Armijo* and *Wolfe* conditions, and if obeyed to them, the convergence will be secured [41, p. 377].

3.2.2. Constrained Optimization

Constrained optimization can fundamentally be solved using the same procedures defined for unconstrained optimization, once the constraints are somehow included in the problem. A C^2 continuous function f is now subject to C^2 equality and inequality constraint functions of the form (3.1d) and (3.1c), where the gradients of the active constraints are linearly independent.

Considering only equality constraints $\mathbf{h}(\mathbf{x})$ first, if their substitution inside the objective function is not possible, which is a common situation, they all can be coupled to the objective function via the use of *Lagrangian function*

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i=1}^{p} \lambda_i h_i(\mathbf{x}) = f(\mathbf{x}) - \boldsymbol{\lambda}^{\mathrm{T}} \mathbf{h}(\mathbf{x}) \,, \tag{3.13}$$

where $\boldsymbol{\lambda}$ is a vector of Lagrange multipliers.

Through the use of Lagrange multipliers, the equality constrained problem is recasted as an unconstrained problem. Now, instead of the original function f, the equality constraint coupled Lagrangian function can be solved by using the same optimality conditions.

Coupling inequality constraints $\mathbf{g}(\mathbf{x})$ is quite more complicated than equality constraints. Unlike equality constraints, inequality constraints do not necessarily have to "bind" the solution at all times, *i.e.* a solution can satisfy the constraint by being arbitrarily less than the equality definition. Or differently interpreted, if an inequality constraint binds the solution, it can be modeled as an equality constraint. This leads to the need of specifying which of the inequality constraints are binding, or *active*, and which of them are not binding, or *inactive* at the solution while searching. More precisely stated, an inequality constraint is either active, by proscribing a region that would behold a minimum if the constraint itself would be absent, or is inactive, by proscribing a region that would not behold the minimum even if the constraint itself would be absent, [47, p. 40]. This corresponds to the *complementary slackness condition*, which explicitly points out that for the inactive case, Lagrange multiplier corresponding the constraint to be zero. If an active constraint has a Lagrange multiplier equal to zero, it is called a *degenerate constraint*. Degeneracy is an important issue as it may lead the algorithms to converge very slowly, or even to fail [41, p. 497]. Another reason of the degeneracy is the linearly dependent, or redundant, constraints [45, p. 466]. If an active constraint can be inferred from another, it is defined as *redundant*.

Implementation of the stated considerations allow the generalization of Lagrange multipliers method. Named after the inventors of this approach, *Karush & Kuhn-Tucker*

(KKT) conditions define the necessary conditions, known as *regularity conditions*, for a solution to be optimal.

In a general case, where nondegenerate both inequality and equality constraints are present for a regular point $\mathbf{x}_*$, there exists a vector of Lagrange multipliers $\boldsymbol{\lambda}_*$ such that

$$\nabla_x \mathcal{L}(\mathrm{x}_*, \boldsymbol{\lambda}_*) = \mathbf{0}\,, \tag{3.14a}$$
$$\boldsymbol{\lambda}_* \geq \mathbf{0}\,, \tag{3.14b}$$
$$\mathbf{h}\,(\mathrm{x}_*) = \mathbf{0}\,, \tag{3.14c}$$
$$\mathbf{g}\,(\mathrm{x}_*) \leq \mathbf{0}\,, \tag{3.14d}$$
$$\boldsymbol{\lambda}_*^{\mathrm{T}} \mathbf{h}\,(\mathrm{x}_*) = \mathbf{0}\,. \tag{3.14e}$$

These are the necessary conditions of first order for optimality. The second order sufficient condition is given by

$$\mathbf{y}^{\mathrm{T}} \nabla_{xx} \mathcal{L}(\mathbf{x}_*, \boldsymbol{\lambda}_*) \mathbf{y} \quad \text{to be positive definite,} \tag{3.14f}$$

where $\mathbf{y}$ is a basis for the null space of the Jacobian matrix of nondegenerate constraints at $\mathbf{x}_*$ [41, p. 506]. This actually corresponds to tangent vectors of the active constraints at a solution $\mathbf{x}_*$ [45, p. 330].

It should be noticed that, the Equation (3.14e) corresponds to the complementary slackness condition. In some sources, the Lagrange multipliers of the inequality constraints are termed as *KKT-multipliers* and denoted with $\boldsymbol{\mu}_*$.

The multipliers essentially give the coefficients of the linear combination between the gradient of the objective function and the gradients of the constraints. They therefore define the "sensitivity" of the objective function value to changes in the constraints. Hence, a λ_i equal to zero would indicate that the constraint present does not have any effect on the solution.

The KKT conditions are solved by checking the solutions of the Lagrangian function given in Formula (3.13). Any solution satisfying these constraints will be a local solution of the problem. Among the solutions, only one will yield the optimal result and that will be selected as the solution. The process of finding which of the constraints are active and then taking the solution that is yielding the optimal solution is quite burdensome. For determining which of the sets are active, several approaches exist in literature, such as active set methods or penalty methods. Nevertheless, in a convex problem, any solution found will automatically correspond to the global minimum. Hence, the if a solution satisfies the constraints, there is not any reason to check other solutions whether they yield a better minimum and therefore, the optimization can be terminated.

It should also be noted that, in case of constrained optimization, feasible search directions are not on line any more, but are rather on curves. This result arises from the surface-like geometric structure of constraints. Therefore, small movement that maintain feasibility are said to be as on *feasible curves*.

3.2.3. Sequential Quadratic Programming

Sequential quadratic programming (SQP) methods are the indispensable methods for solving nonlinearly constrained problems, and they exhibit their strengths when significant nonlinearities are present in the constraints. The SQP methods, fundamentally, generalize the Newton's method to constrained optimization, via the use of KKT conditions. They model the objective function, as a linearly constrained quadratic function and use the minimizer of the quadratic function to determine a new iteration of the solution.

For an objective function defined in Formula (3.1), the Lagrangian function can be given as

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{c}(\mathbf{x})\,, \tag{3.15}$$

where $\mathbf{c}(\mathbf{x})$ is a vector of equality constraints obtained from KKT conditions. The vector is in the same length with the summation of number of equality and inequality constraints, that means it has $m + p$ elements in total, and all of the active constraints are linearly independent.

The first order optimality condition can be given as

$$\nabla\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = 0\,, \tag{3.16}$$

or in open form

$$\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}_k, \boldsymbol{\lambda}_k) = \nabla f(\mathbf{x}_k) - \nabla\mathbf{c}\,(\mathbf{x}_k)\boldsymbol{\lambda}_k = \mathbf{0}\,, \tag{3.17a}$$

$$\nabla_{\boldsymbol{\lambda}}\mathcal{L}(\mathbf{x}_k, \boldsymbol{\lambda}_k) = -\mathbf{c}\,(\mathbf{x}_k) = \mathbf{0}\,. \tag{3.17b}$$

$\nabla\mathbf{c}$ represents the transpose of the Jacobian of each constraint $c_i,\ i \in (1, ..., m + p)$.

The Newton's method, given in its basic form in Equation (3.8b) can be used for iteratively solving the Equation (3.16) or (3.17):

$$\begin{pmatrix} \mathbf{x}_{k+1} \\ \boldsymbol{\lambda}_{k+1} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_k \\ \boldsymbol{\lambda}_k \end{pmatrix} + \begin{pmatrix} \mathbf{p}_k \\ \boldsymbol{\nu}_k \end{pmatrix} \tag{3.18}$$

However, in order to be able solve the equation (3.18), $\mathbf{p}_k$ and $\boldsymbol{\nu}_k$ must be known. These can be obtained from the following linear system:

$$\nabla^2\mathcal{L}(\mathbf{x}_k, \boldsymbol{\lambda}_k) \begin{pmatrix} \mathbf{p}_k \\ \boldsymbol{\nu}_k \end{pmatrix} = -\nabla\mathcal{L}(\mathbf{x}_k, \boldsymbol{\lambda}_k)\,. \tag{3.19}$$

Or, in open form expressed as

$$\begin{pmatrix} \nabla^2_{\mathbf{xx}}\mathcal{L}(\mathbf{x}_k, \boldsymbol{\lambda}_k) & -\nabla\mathbf{c}(\mathbf{x}_k) \\ -\nabla\mathbf{c}\,(\mathbf{x}_k)^{\mathrm{T}} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{p}_k \\ \boldsymbol{\nu}_k \end{pmatrix} = \begin{pmatrix} -\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}_k, \boldsymbol{\lambda}_k) \\ \mathbf{c}(\mathbf{x}_k) \end{pmatrix}\,. \tag{3.20}$$

The equation (3.20) corresponds to the following quadratic optimization problem:

$$\min_{\mathbf{p}} q(\mathbf{p}) = \tfrac{1}{2}\mathbf{p}^{\mathrm{T}}[\nabla^2_{\mathbf{xx}}\mathcal{L}(\mathbf{x}_k, \boldsymbol{\lambda}_k)]\mathbf{p} + \mathbf{p}^{\mathrm{T}}[\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}_k, \boldsymbol{\lambda}_k)]\,, \tag{3.21a}$$

$$\text{subject to } [\nabla \mathbf{c}\ (\mathbf{x}_k)]^{\mathrm{T}}\mathbf{p} + \mathbf{c}(\mathbf{x}_k) = 0\,. \tag{3.21b}$$

It can be realized that the equation (3.21a) corresponds to Taylor approximation of the Lagrangian at $(\mathbf{x}_k, \boldsymbol{\lambda}_k)$, and the equation (3.21b) corresponds to the linear approximation of the constraints at $\mathbf{c}(\mathbf{x}_k + \mathbf{p})$. This an expected situation of applying KKT conditions and Newton's method.

The SQP methods exhibit the same characteristic properties as the Newton's method. The convergence properties are highly dependent on the initial estimation done. If the initial estimation $\mathbf{x}_0, \boldsymbol{\lambda}_0$ is close enough to the solution $\mathbf{x}_*, \boldsymbol{\lambda}_*$, than it will deliver quadratic convergence. For globalization, either merit functions or trust region methods can be used. Merit functions include terms for the assessment of the candidate step width, yielding a "better" estimate of the solution and feasibility-degree regarding constraints. These terms penalize undesired steps via the penalty methods or barrier methods and increase the value of the merit function. Subsequently, a step that returns the smallest result is taken. Analogous conditions to the conditions for unconstrained problems are subject to merit functions. Algorithms depending on merit functions, however, may undergo an undesirable phenomenon called *Maratos effect*, which may invalidate the superlinear convergence in the absence of precautions [45, p. 440].

The computational cost, also as in the Newton's method, is dominated by the computation of Hessian matrix. Therefore, it is common for many SQP methods to use quasi-Newton methods, which are categorized as *full* quasi-Newton methods and *reduced* quasi-Newton methods. However, regard must be payed to the choice of the Hessian approximation, as it is in core of the solution process and the convergence properties of the algorithm are influenced by it [41, p. 576].

In this section the solution methodology and the fundamental theorems behind the SQP-methods have been introduced. These serve as the foundation and reflect the main considerations behind all SQP algorithms. The theory of optimization is immense, and there are other crucial issues such as ill-conditioning of the Hessian, which can, for example, occur in the utilization of penalty functions, or as a natural result of an active constraint binding a solution [41, p. 618]. Furthermore, in case of sparse Hessian or sparse constraint matrices, which is a typical case for moderate to large problems, there are some special techniques called *sparse matrix techniques* [41, p. 366]. These techniques reduce the storage and the computational cost of using Newton's method. Apart from the general issues, even sole SQP family is a vast area of research and there is a huge literature about these methods. The context of this work does not include improving the algorithmic details of the solver used for the trajectory optimization. Hence, the investigation of the optimization theory and SQP methods will be finished here. For a comprehensive survey of SQP-methods readers are referred to Chapter 18 in [45] or to [48]. The used SQP algorithm is presented in Section 6.2.

The flowchart a SQP-algorithm with a quasi-Newton method is illustrated in Figure 3.4. The quadratic subproblem controlling the steps of the optimization can be seen.

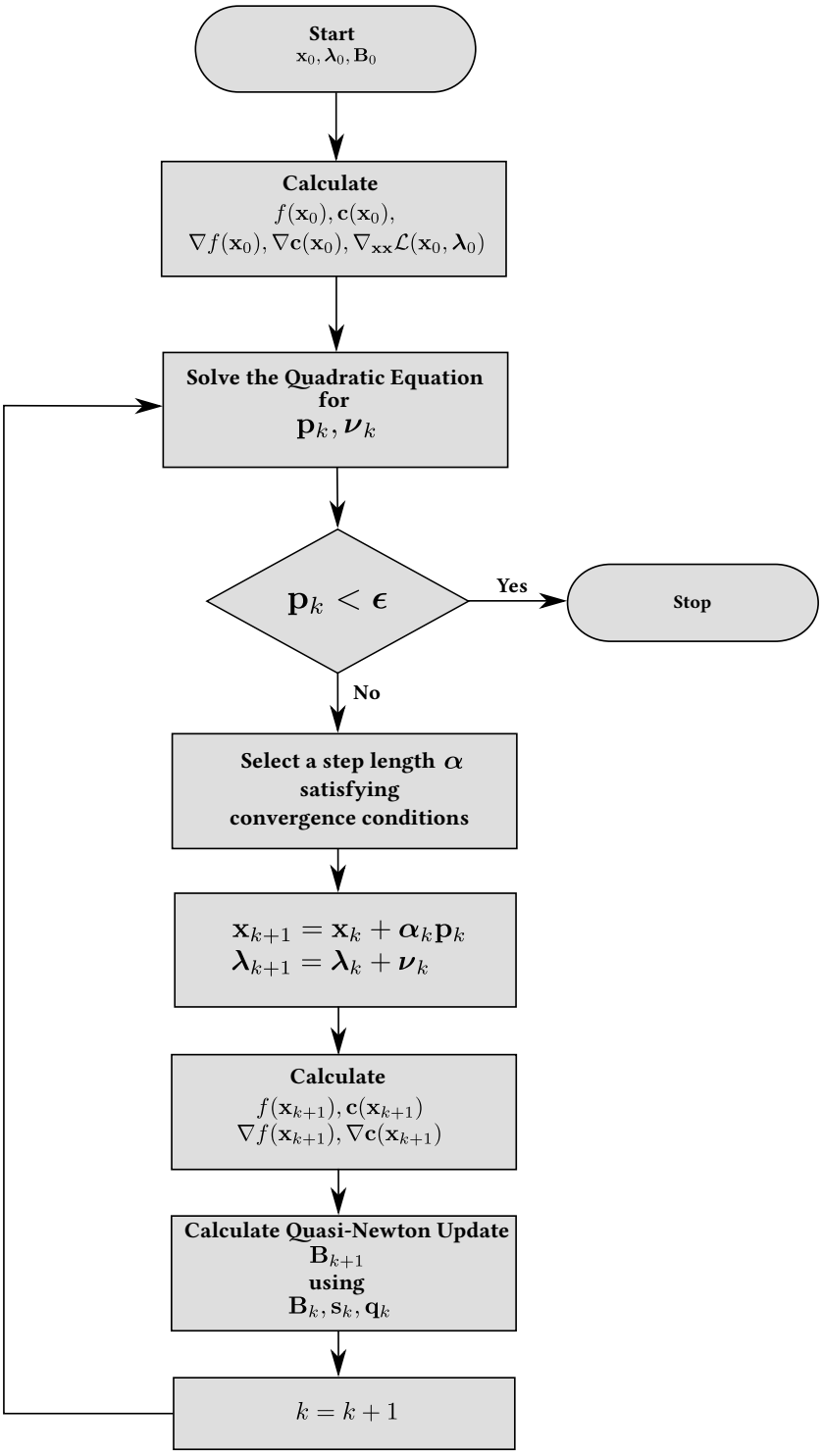


Figure 3.4: Flowchart of a SQP algorithm with quasi-Newton update.

4. Trajectory Planning as an Optimization Problem

In Chapter 2, various motion planning methods were reviewed and the reasons for the utilization of optimization based methods was mentioned. In Chapter 3 the theoretical backgrounds and essential information about these methods were given. This chapter focuses on the formulation of motion planning as a constrained optimization problem.

The chapter starts with introducing formulation for the most simple case, namely where the only task is to plan an optimal trajectory along an obstacle free, structured environment. Structure of the cost function and the required constraints to maintain a feasible trajectory planning are given in this chapter. These will later serve as the basis for more complex scenarios, where other traffic participants are present.

Large extend of the work presented in this chapter has already been published in [34].

4.1. Problem Definition and Formalization

The first step at establishing any type of optimization problem is the identification of the optimization parameters and their underlying dimensionality. If the vehicle is reduced to a point, where the point corresponds to the center of the rear axle, its motion can be represented by the position of that point along time axis. Defining such a point as *node* $\mathbf{x}$, any node in global coordinate system can be represented by two variables: through x and y coordinates. Hence, the dimensionality of finding a single position is 2, or in other words, each node introduces two optimization parameters. The vehicle motion is conventionally planned for a certain finite time in the future, which commonly is referred as *time-horizon*, or *horizon* T for short. It is beneficial to select the horizon as large as possible, however, the computation time is a bounding factor and therefore it cannot be selected as arbitrarily large. While the range of the horizon may variate, reasonable values are in between 8-16s.

Once the parameters and the dimensionality of the optimization problem is introduced, the function that adjusts the parameters under consideration of certain constraints must be given. Subsequently, in order to be able to compute these formulations in digital computers, obtained equations for continous case must be transformed into their discrete counterparts. The respecting objective and constraint functions and their discretization are given in the subsections below.

4.1.1. Objective Function

The objective function is the core of an optimization process. It is the function that sets the properties of ideal trajectory up and penalizes the deviations from it with increasing cost values. Certainly, an ideal trajectory has to reflect the desired profile of a motion, such as minimization of the lateral distance from driving corridor centerline, or driving with the reference set-speed. On the other hand, it should be remarked that the fulfillment of requirements may in some cases be impractical by means of comfort, or even energy consumption. For example, driving on a road with hairpin turns at the same speed as if it were a straight road will decrease the ride comfort. Or as a further example, if a trajectory contains rapid accelerations and decelerations, corresponding fuel consumption will be high, and the comfort will be low. To sum up, for a trajectory to be optimal, besides the terms that comprise the main requirements set on motion, also terms that smooth the trajectory have to be implemented to the objective function.

In this respect, an optimal trajectory at time t_0 can be obtained by minimizing the following cost functional:

$$\mathcal{J}[\mathbf{x}(t)] = \int_{t_0}^{t_0+T} L(\mathbf{x}, \dot{\mathbf{x}}, \ddot{\mathbf{x}}, \dddot{\mathbf{x}})\,\mathrm{d}t\,, \tag{4.1a}$$

where L is given by

$$L = j_{\text{offset}} + j_{\text{velocity}} + j_{\text{acceleration}} + j_{\text{jerk}}\,. \tag{4.1b}$$

The individual summands of the integrand L respectively represent the offset from the imaginary centerline, the deviation from desired velocity, and terms that dampens excessive acceleration and jerk values.

The offset term is obtained by using the squared normal distance from the road-centerline d_{center} and then by multiplying it with a relative weighting factor, as indicated below:

$$j_{\text{offset}}(\mathbf{x}(t)) = w_{\text{offset}} |d_{\text{center}}(\mathbf{x}(t))|^2. \tag{4.2}$$

Evidently, there is not any such line on the roadways and this merely represents a virtual line. Another option would be to use the sum of signed distances from the right and left corridor bounds, instead of the d_{center}. This would be the only possible choice for a simultaneous mapping and driving application. However, if the map of the environment would be available prior to driving, creating such centerlines and adding them into map-data would be beneficent. During planning, this would result in deriving distance from only one line, which thus reduces the computational costs of a single query. Considering the objective function is evaluated multiple times during planning, this could result with notably shorter computation times.

The second term penalizes any deviation of velocity from the reference value $\mathbf{v}_{\text{des}}$, and is given by:

$$j_{\text{velocity}}\left(\mathbf{x}(t)\right) = w_{\text{velocity}} \left|\mathbf{v}_{\text{des}}(\mathbf{x}(t)) - \dot{\mathbf{x}}(t)\right|^2. \tag{4.3}$$

The reference velocity is always in tangential direction to the road boundaries, with its magnitude being equal to the desired travel speed. The direction of the reference velocity for an arbitrary node is given in Figure 4.1. As an intrinsic result of the change of the road geometry in $(x \times y)$ coordinates, the tangent vector also generally points to disparate directions throughout the road. Hence, the reference velocity can only be defined by transforming unit tangential vector to its x- and y-components and then multiplying it with the scalar reference speed s_{des} value.

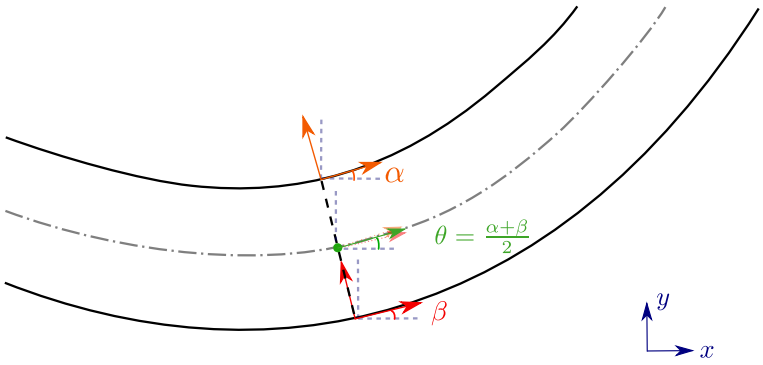


Figure 4.1: Normal vector of reference velocity.

There are several alternative ways to define the reference velocity. The most basic option would be to set it simply equal to the desired speed, without explicitly defining its x- and y-components. However, such a definition does not discern between forwards and backwards driving. Both of them return same cost. Such a case obviously represents an undesirable case and is far away from being a feasible alternative.

Both of the introduced terms until now serve the vehicle to follow a given trajectory. As discussed, smoothness can be maintained by the following terms:

$$j_{\text{acceleration}}\left(\mathbf{x}(t)\right) = w_{\text{acceleration}} \left|\ddot{\mathbf{x}}(t)\right|^2, \tag{4.4}$$

and

$$j_{\text{jerk}}\left(\mathbf{x}(t)\right) = w_{\text{jerk}} \left|\dddot{\mathbf{x}}(t)\right|^2. \tag{4.5}$$

The selection of the objective function not only has a vast impact on the result, but it also constitutes the form of the optimization problem and plays a key role on the selection of the optimization method. In section 3.1, it was clarified that local and gradient based solvers are able to return results within a short amount of time and a real time optimization implementation can achieved only by their utilization. It was also mentioned that they only can return a local minimum, and this might be in some cases a

real consideration. Therefore, the cost function has to be selected in such that it does not contain any stationary point, to which, if encountered, the solver would get stuck to.

It can be noticed that there are multiple objectives present, some of which having even contradicting inclinations on the selection of optimization parameters. This is the form of a typical multicriteria optimization, as introduced in Section 3.1, on Page 20. An aggregation of the individual summands can be realized by introducing weighting factors, that define the relative impact of each parameter. It is, however, crucial to use relative factors, and to abstain from using absolute factors. Although the use of absolute factors is merely like multiplying the relative factors with some scalars, practical experience have shown that in medium to large problems this caused the gradient computation to fail, probably because of numerical effects.

The choice of weighting parameters plays a key role on the profile of generated trajectory. Their selection was done in a similar way like finding weighting factors of any other engineering problem. As a first step, they are selected such that each parameter has roughly the same order of magnitude. Afterwards, they are multiplied with scalars according to their relative importance. The final step was to use experimentation, and tuning the parameters with the insights obtained from overshoot and oscillation in control theory.

A further issue to notice is the quadratic from of the cost functional. As mentioned in Chapter 3, objective functions of quadratic form are convex, and this makes the local-gradient based optimization algorithms especially attractive for such problems. Any local minimizer of such a function will correspond to the global minimum.

4.1.2. Constraints

In order to fit the optimization parameters into a range that yields feasible trajectories, the objective function must be subject to certain constraints. In gradient-based optimization, besides the cost function, the constraints must be continuous as well. Constraints in motion planning are divided into two categories: *internal constraints*, that arise due to the physical limits of the autonomous agent, and *external constraints*, that result from the environment of the agent. Although the primary interest of the thesis is on external constraints, the internal constraints will be briefly reviewed before dealing with the external constraints.

Vehicle motion is limited by the kinematics of the steering geometry and by the forces that the engine can deliver, or in other words the tires can transmit to the road surface. Typically, at low speeds restrictions due to bounds of turning radius become more prevalent. It is common to use the curvature, which is the inverse of turning radius, while defining cornering. This was previously illustrated on Figure 2.1, and was denoted

with the symbol κ. In Cartesian coordinates, curvature at any instant can be extracted from velocity and acceleration information using the formula below:

$$\kappa(t) = \frac{\dot{x}(t)\,\ddot{y}(t) - \dot{y}(t)\,\ddot{x}(t)}{\sqrt[3]{\dot{x}(t)^2 + \dot{y}(t)^2}}, \tag{4.6a}$$

where the κ is bounded by a maximum value:

$$|\kappa(t)| < \kappa_{\max}\,. \tag{4.6b}$$

Such a symmetric, state-independent bound $\kappa_{\max}$ corresponds to the simplest type of actuator limit. For an ordinary sedan vehicle, turning radius is generally between 4–5 meters.

At high speeds, due to the increasing aerodynamic forces and rolling resistance of the tire, the longitudinal acceleration limit is lower than it is at lower speeds. Moreover, the friction limit of the tire plays more decisive role on the feasibility. This circumstance is best described by the circle of forces and can very roughly be formulated as

$$\|\ddot{\mathbf{x}}(t)\|^2 < {a_{\max}}^2, \tag{4.7}$$

where $a_{\max}$ represents the radius of the circle. A better limitation could be given by separating longitudinal acceleration from lateral, and splitting the longitudinal acceleration limits into acceleration and deceleration. If dealing with tanker trucks, it may also be necessary to limit jerk in the same way as for acceleration. Jerk values can be crucial for all objects which do not have a stable mass center. The dynamics of tanker trucks are closely related to the fill rate of the liquid they are transporting, and if not fully loaded, changes in acceleration values play a deciding role.

Travel speed is also bounded typically by the regulations of the environment. Therefore, they fit under the group of external constraints. For a speed limit $s_{\max}$ specified by the regulations, the inequality below must hold:

$$\|\dot{\mathbf{x}}(t)\|^2 < s_{\max}\,. \tag{4.8}$$

The major external constraint in the absence of obstacles is undoubtedly the constraint, that keeps the vehicle inside the driving corridor. This can be maintained by using the signed distance. Signed distance of a point with respect to a line is positive on one side of the line, and negative at other side. A contour plot of an arbitrary line is given in Figure 4.2. Reddish colors indicate positive values and the bluish ones negative values. If the signed distance of a point is taken with respect to the right and left bounds of the road, the multiplication of the distances will be negative inside the driving corridor, and be positive outside. Capitalizing on that, the vehicle can be kept inside the driving corridor by constraining the centerpoint to have a negative value.

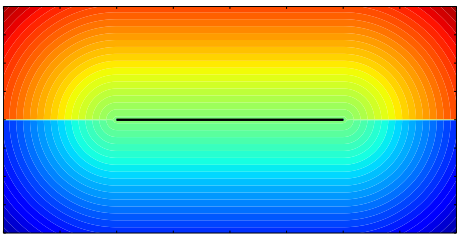


Figure 4.2: Signed distance function of an arbitrary line. Smoothness at the both edges should be remarked. Only the normals at the both tips of the reference line are discontinuous.

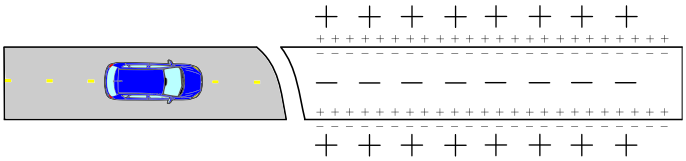


Figure 4.3: Roadside boundary constraint.

However, there is another issue to be taken into consideration. The vehicle was reduced to a single point that corresponds to the rear axle centerpoint. Of course, checking barely that point is not sufficient as this would set only that point to be inside the driving corridor, and not the whole vehicle body or the maneuver area. Two remedies for that will be introduced here. One way is to calculate the position of the edges of the vehicle and perform the check for those four corner points instead of the axle centerpoint. Vehicle corners to be regarded are illustrated in Figure 4.4.

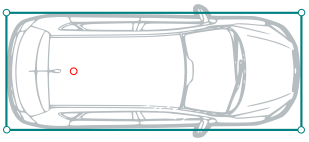


Figure 4.4: Vehicle corners to be regarded for the driving corridor constraint.

Although this might be the felicitous solution, checking four points instead one would increase the computational costs. A practical way is to benefit from $\mathbb{R}^2$ workspace and grow the configuration space by a certain length r up. Constraint formulation c_{corridor} in this case for the centerpoint $\mathbf{x}$ can be given as

$$c_{\text{corridor}}(\mathbf{x}) = \left(d_{\text{right}}(\mathbf{x}) - r\right) \cdot \left(d_{\text{left}}(\mathbf{x}) + r\right), \tag{4.9a}$$

where

$$c_{\text{corridor}}(\mathbf{x}) < 0, \forall \mathbf{x} \in \mathcal{W}_{\text{corridor}}\,. \tag{4.9b}$$

d_{right} and d_{left} indicate the signed normal distance from the right and left corridor bounds, respectively. The minimum value of r depends on the geometry of the vehicle.

A proper selection would be greater than the diagonal length from axle-centerpoint to the furthest cornerpoint. In fact, this corresponds to decomposing vehicle shape into a big circle. However if the vehicle is passing through a narrow corridor in parallel, then the half of the vehicle width can also be taken. Such parallel motions make the method to become quite more practical. A further way is to use circle decomposition of the vehicle, as presented in [12].

These were the fundamental constraints that maintain a feasible reference trajectory. Any instance of a planned trajectory must observe these limits.

4.1.3. Discretization

Once the formal representations of the functionals are clarified, due to reason mentioned in the beginning of the current section, they have to be transformed to functions via the use of finite differences. The integral was defined over time t and now it will be discretized into equally spaced intervals called *timesteps*. In this case, the horizon T can be divided into $N \in \mathbb{N}$ timesteps and the trajectory will be approximated via them. If the trajectory would be approximated by step-widths of Δt, a formulation for N can be given as:

$$N = \frac{T}{\Delta t}, \tag{4.10}$$

and any time t_i after t_0 would be

$$t_i = t_0 + i\Delta t, \quad 0 \leq i < N. \tag{4.11}$$

The time derivatives of the trajectory nodes can be approximated by a various finite difference formulations. Considering efficiency and computational cost, forwards and backwards differences have found more application in pratice. Furthermore, unlike the central differences, they do not introduce any smoothing effect on the obtained derivatives, which could lead to deceptive results. Although backwards and forwards differences essentially yield the same result, the timestep for which they find the derivative varies. For sense of simplicity in indexing, backwards differences are used. It should be remarked that, in order to obtain the jerk value at a node, the positions of three nodes in prior to it are needed. Time derivatives obtained by utilizing backwards difference of trajectory nodes $\mathbf{x}_i$ can be given as:

$$\dot{\mathbf{x}}(t_i) \approx \mathbf{x}_{\mathrm{d}} = \frac{\mathbf{x}_i - \mathbf{x}_{i-1}}{\Delta t}, \tag{4.12a}$$

$$\ddot{\mathbf{x}}(t_i) \approx \mathbf{x}_{\mathrm{dd}} = \frac{\mathbf{x}_i - 2\mathbf{x}_{i-1} + \mathbf{x}_{i-2}}{(\Delta t)^2}, \tag{4.12b}$$

$$\dddot{\mathbf{x}}(t_i) \approx \mathbf{x}_{\mathrm{ddd}} = \frac{\mathbf{x}_i - 3\mathbf{x}_{i-1} + 3\mathbf{x}_{i-2} - \mathbf{x}_{i-3}}{(\Delta t)^3}. \tag{4.12c}$$

In this case, the integral given in equation 4.1a will transform into a finite sum:

$$\mathcal{J}[\mathbf{x}(t)] \approx \mathcal{J}_{\mathrm{d}}(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{N-1}) = \sum_{i=3}^{N-1} L(t_i, x_i, x_{\mathrm{d}}, x_{\mathrm{dd}}, x_{\mathrm{ddd}})\Delta t\,. \tag{4.13}$$

Now, the equation presented above can be minimized with respect to the parameters $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{N-1}$. Each of these parameters consists of x- and y-components and are commonly referred as *trajectory support points*.

Considering the quadratic structure of the cost function and the nonlinear structure of the constraints, the problem corresponds to a nonlinearly constrained optimization problem. As introduced in Chapter 3, the SQP methods among other local-gradient based algorithms are renowned as powerful solvers for this type of problems.

4.2. Initialization

Local optimization algorithms require an initial guess to start the optimization. The quality of the solution depends heavily on the initial guess. As indicated in Chapter 3, even the Newton's method converges quadratically only if the initial guess is close enough to the solution.

Initial guess plays a key role not only on the speed of convergence, but also on whether the algorithm convergence at all. There is not any proof convergence if the initial guess is not in the feasible solution set S. Although, practical experience shows that the algorithms that rely on the true Hessian mostly tend to find a feasible solution after a certain time, even if the initialization is not inside S. Nevertheless, it should be refrained from bad initializations and it must be striven for performing feasible and solution-proximate guesses.

Starting with the simplest case, in which the task is to perform initialization for a vehicle heading on $+x$ direction, an initial guess can be made by multiplying the desired travel speed with the respective timesteps and step-width, and eventually summing it up with the initial position, *i.e.*

$$\mathbf{x}_i = [x_i,\ y_i] = [x_0 + s_{\mathrm{des}}\, i\Delta t,\ y_0]. \tag{4.14}$$

Almost the same formulation can be used for any straight motion on Cartesian coordinates. However, when the vehicle has to drive on a curved road, such a formulation will not be tractable any more. In that case, coordinates must be defined with respect to the geometry of the street. Hopefully, such a reference exists in differential geometry and is called the Frenet frame [49]. Coordinates in Frenet frame, as illustrated in Figure 4.5, are defined by taking the signed distance of the closest normal d to the curve, and the arc length l of the curve until the point to which the normal corresponds. By exploiting this frame system, initializations invariant to street geometry can be performed.

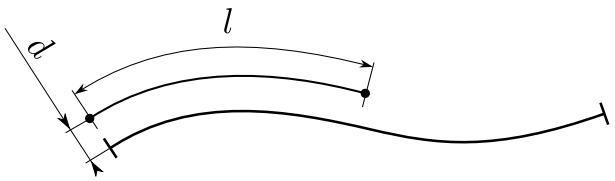


Figure 4.5: Frenet coordinates with respect to a reference curve.

The first is to find coordinates in Frenet frame by exploiting the formulation given in Equation 4.14:

$$[l_i,\ d_i] = [l_0 + s_{\text{des}}\, i\, \Delta t,\ d_{\text{reference}}] \tag{4.15}$$

Afterwards, the obtained coordinates are transformed to $(x \times y)$ coordinate system, to which the optimization is defined on:

$$\mathbf{x}_i = [x_i,\ y_i] \leftarrow [l_i\,, d_i]\,. \tag{4.16}$$

From now on, transformation from $(x \times y)$ coordinates to $(l \times d)$ coordinates will be defined as *forwards* transformation, and the vice-versa as *backwards* transformation.

The selection of the reference normal distance $d_{\text{reference}}$ depends on the reference line selected, *i.e.* if the centerline of the road is selected as the reference line, $d_{\text{reference}}$ can be taken equal to 0. On the other hand, if the right or left bound of the road is used, than $d_{\text{reference}}$ must be taken equal to the half of the lane width.

It should be remarked that, although the coordinate system was changed, the speed used to calculate the displacement s_{des} has remained the same. As illustrated in Figure 4.6 below, displacement calculation in the vicinity of roadside boundaries leads to fallacious results. However, considering that the vehicle is expected to center the lane, an appropriate initialization would minimally suffer from this effect, eventually making this concern to become insignificant.

The initialization model represented by the Equation 4.15 does not incorporate the effect of initial acceleration and jerk values. Certainly, if their corresponding values are high, and the horizon is limited to a few seconds, this would lead to a bad initialization. Nevertheless, the intended horizons are typically greater than 8-10 seconds. In such a case, even though the initial values of acceleration and jerk may be high, if the initial speed does not deviate from the desired travel speed remarkably, the initialization would be *successful* in overall. Furthermore, if the effect of acceleration is not included into the predicted trajectory calculation, the acceleration values throughout the motion would only vary in consequence of the change in the road curvature. If the vehicle is not heading at high speeds on sharp turns, the lateral acceleration values would also be typically below the defined constraint limits. Albeit this might seem like quite an optimistic approach, even if the acceleration values exceed the limits, the optimizer will presumably find, may be not very optimal but, a feasible solution. An enhancement on the initialization will however be presented in Section 5.2.3, and another way of initialization will be presented in Chapter 8, while proposing future work.

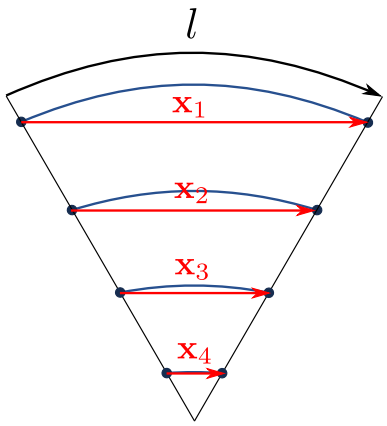


Figure 4.6: Comparison of displacement measurement in Frenet coordinates with Cartesian coordinates. Red arrows indicate the measurement in Cartesian and the black arrows in Frenet coordinates. All of the black arrows represent the same displacement value l, whereas each of the red arrows represent distinct values.

The constraint on steering angle will be inherently satisfied by any initialization done, as roads are constructed respecting the physical limits of vehicles. In case the corridor boundaries are generated by the autonomous vehicles perception system simultaneously, then a plausibility test for the generated road geometry must be performed and unsuitable road geometries arising from faults of the perception system must thereby be filtered out. The other constraint that keeps the vehicle inside the road boundaries will also be satisfied at any instance if $d_{\text{reference}}$ is adjusted such that it corresponds to the center of the road. Hence, it will also not be a matter of concern.

5. Kinodynamic Optimal Trajectory Planning

An autonomous vehicle typically operates in an environment which comprises dynamic obstacles. This requires the optimal planning method presented in Chapter 4 to be extended in such a way that it delivers optimal trajectories even if the path of the vehicle is occluded by other vehicles. The intention of this chapter is to give the necessary formalizations that yield a collision free optimal trajectory in the presence of velocity obstacles. The presented methods in principle are also applicable to static obstacles, as they represent a special case of velocity obstacles.

The chapter starts with introducing combinatorial problems encountered in roadways and continues with presenting a method for the identification of all feasible combinations. Subsequently, constraints that yield combination restrained collision-free trajectories are introduced. As these additional constraints set on motion must also be satisfied by the initial guess, the method presented for initialization must be extended appropriately. Finally, considering the computational burden resulting from the multi-combinatorial structure of the environment, the so called path-velocity decomposition is described together with its implementation.

5.1. Combinatorial Problems in Roadways

Driving inherently involves decision making about various combinatorial options. The most basic combinatorial problem to be solved starts even prior to the start of motion, with route planning. Considering the case where a driver needs to reach B from A and assuming there is a variety of occasionally intersecting roads leading to the destination, driver selects a route among the alternatives according to a certain criteria, such as travel time, path length etc. For that purpose, the driver evaluates several alternatives, and in case combines several road segments to obtain the optimal path. Although the combinatorial problem seems to be settled, in fact, it starts once the vehicle hits the road.

Any obstacle on roadways, static or dynamic, create combinatorial possibilities from which the driver has to choose the most favorable one. In its simplest form, a road that is partly blocked by a static obstacle can be analyzed. Such a situation, where a tree

located at about the center of the road is illustrated in Figure 5.1. The driver in that case must decide either to drive on the right or on the left side of the obstacle.

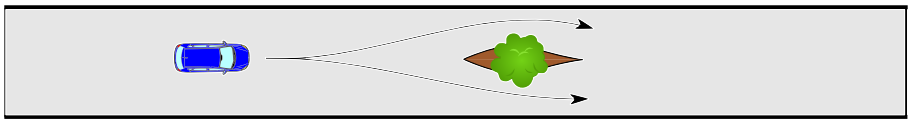


Figure 5.1: Combinatorial problem in presence of a static obstacle.

Or as a further example, a situation from dynamic street scenarios, such as the one illustrated in Figure 5.2, can be considered. As illustrated in the figure, assuming that there is not any traffic sign controlling the merge-flow, the ego vehicle, represented in blue color, has to decide in which sequence it will enter the intersection. That is, it can either yield to other vehicles, or accelerate and become the leading vehicle, or take a position in between them.

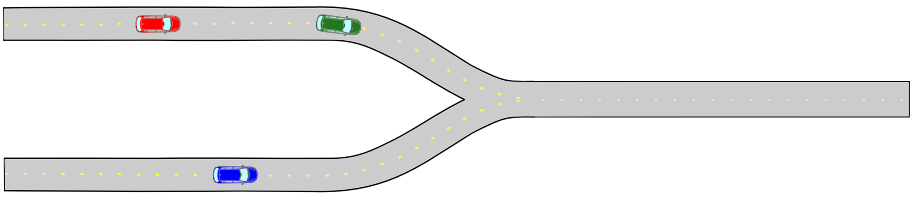


Figure 5.2: A basic combinatorial problem in dynamic street scenarios.

When treated as an extremum problem, the presence of multiple combinations would generate multiple local minima, and the objective function in overall would look like one given in the Figure 3.2. However, due to the fact that some positions that the vehicle can be located on are already blocked by their vehicle widths and and by a safety distance, some regions of the figure would be restricted. Considering the merge situation illustrated in Figure 5.2 as an example, the cost function for the combination "driving in between both vehicles" could be depicted as in Figure 5.3.

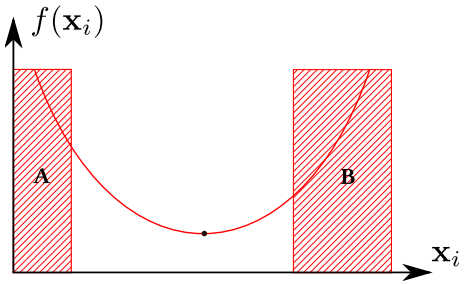


Figure 5.3: Illustration of the objective function for the combination of 'merging in between vehicles' in the merge scenario given in Figure 5.2. **A** represents the region restricted by the red vehicle and **B** by the green vehicle.

Drivers unwittingly make lots of such decisions while driving. These are however not always *correct*, in sense they select the most favorable combination. An autonomous

vehicle optimal trajectory planner in contrast has always to pick up the most favorable option, *i.e.* the global minimum, and plan its trajectory accordingly. Finding the global minimum problem in such a case would typically require the utilization of global optimization algorithms. However, the drawbacks of global optimization algorithms were mentioned in Chapter 3. In Chapter 4, it was shown that the ordinary trajectory problem has a nonlinear-convex structure and is best solved by local-gradient based algorithms. Apparently, if the problem could be decomposed into individual parts representing distinct combinatorial possibilities, each of the combinations would have a convex cost function and their minima would eventually become attainable by local-gradient based algorithms. Naturally, each of the separate combinations would yield a local minimum, whereas one of them would correspond to the global minimum. The approach in this work will derive constraints that confine the search space to each of the feasible combinations and will afterwards perform constrained local optimization.

5.2. Integrating Combinatorial Reasoning into Trajectory Planning Problem

How obstacles on roadways create a variety of alternative feasible trajectories and thereby pose combinatorial problems on motion planners was introduced in the preceding section. Focus will now be given on their integration into the trajectory planning problem. Necessary formulations will be developed in three steps. The first step would be to determine all of the combinatorial options. Once they are found, generalization about their imposition onto the optimization problem will be given as the second step. Finally, as a requirement of exploiting local optimization algorithms, a method for an initialization obeying the combinatorial constraints will be presented.

5.2.1. Extraction of Combinatorial Options

A practical and complete representation of available combinatorial options plays a key role on the selection of the most favorable combination. Because optimization is an expensive process, optimal trajectory computation for any combination should be done once the combination is found to be feasible and is distinct from other combinations listed. Hence, the representation must not only be complete by comprising all of the combinations, but also must be freed from any redundancy while maintaining feasibility.

A reasonable approach is to divide the extraction problem into two subsequent steps. The objective of the first step would be to list all of the combinations up, while avoiding redundancy. As the second step, by interpreting feasibility conditions as a filter, the combinations found can be passed through it and infeasible ones can thereby be eliminated. In fact, this merely corresponds to applying feasibility constraints on the combinations found, *i.e.* if a combination satisfies the constraints it will be retained, else,

it will be skipped. Hence, the inspection on the feasibility of a combinatorial option can be done once the constraints are formalized.

Any vehicle on a roadway will introduce two options for the ego vehicle while ordering, or as will be called from now on, while *sequencing*. The ego vehicle can be either sequenced before, or after a reference vehicle on its path. It is advantageous to describe combinations of sequencing by binary numbers. In the former case, *i.e.* driving in front of the reference vehicle, the binary representation can be given as '1', and in the latter case, *i.e.* while driving behind of the vehicle, it can be given as '0'. Neglecting feasibility, in the presence of v number of vehicles, the total number of possible combinations C_{all} can be given as:

$$C_{\text{all}} = 2^v. \tag{5.1}$$

Each of the combinations can be obtained by the binary representation of all of the numbers until 2^v. Considering the case illustrated in Figure 5.2, the presence of 2 vehicles lead to $2^2 = 4$ distinct combinations. And the possible combinations are $(0)_2$ = '00', $(1)_2$ = '01', $(2)_2$ = '10', $(3)_2$ = '11', where each digit represents the relative binary position to a distinct vehicle. Hence, if the vehicles would be listed up in a fixed list, all of the possible combinations could be obtained.

5.2.2. Formalization of Constraints

Formalization of combinatorial constraints and filtering infeasible combinations out is not as easy as listing up the possible ones. Because optimization is performed once the combination is found to be feasible, feasibility must be checked prior to the start of optimization. Feasibility of a combination can be investigated by checking whether it satisfies the introduced constraints or not. Therefore, focus will now be given on the formalization of constraints.

In order to facilitate formalization, the previous example can be considered. Assuming that the first digit represents the relative position to the red vehicle and the second digit represents the relative position to the green vehicle, it can be shown that the ego vehicle cannot be in front of the green vehicle, but also behind of the red vehicle at the same time. This indicates that the combination '01' is infeasible. That was the most basic infeasibility to determine. However, there are other factors bounding the motion range of the vehicle and their investigation is quite elaborate. These may arise both from the internal and the external constraints, for which the speed and the acceleration may be a limiting factor due to the geometry of the road. Considering the example illustrated, the path length until the intersection may be so long that even a very fast and agile vehicle would not manage to reach the intersection before the other vehicles and become the lead vehicle. As a result, the sequence '11' may become invalid. In order to analyze alike cases, the vehicle positions must not be only defined in Frenet frame, but they must also be referenced with respect to a common basis. For practical reasons the Frenet frame of the ego vehicle is taken as the reference. In this case, as

illustrated in Figure 5.4, the position of any other vehicle will be given by subtracting the difference in arc-lengths Δl from the vehicle's l-coordinate

$$l_{\text{ref\_vehicle}} = l_{\text{ref\_vehicle,self}} - \Delta l, \tag{5.2}$$

where the Δl is obtained by subtracting ego vehicle's reference line length until the intersection ($l_{\text{path,ego}}$) from that of the considered vehicle ($l_{\text{path,ref\_vehicle}}$) :

$$\Delta l = l_{\text{path,ego}} - l_{\text{path,ref\_vehicle}}\,. \tag{5.3}$$

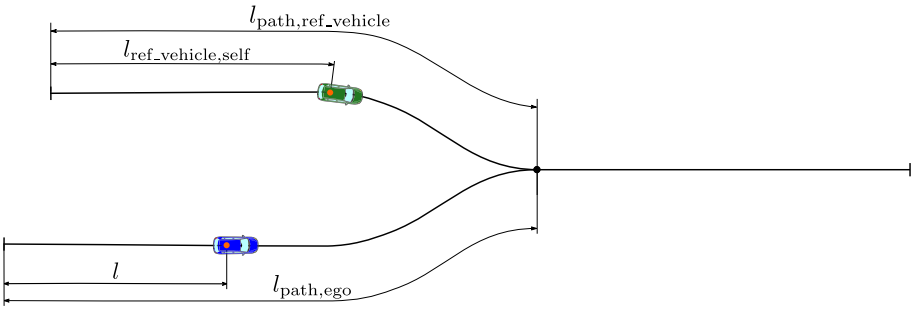


Figure 5.4: Referencing path-coordinates of other vehicle with respect to ego vehicle.

Once the vehicles are referenced on a common basis, their motion can be considered on *path-time* plane, whose abscissa values represent time of motion, and the ordinate values represent the arc-length from the beginning of the reference line to the rear axle center of the vehicle [50]. Although such a representation neglects the information about d-coordinates, it provides an overview to the obstacles present and ameliorates the investigation of alternative combinations. When motion of vehicles are plotted on path-time diagram, as they were modeled as point-objects, their motion will correspond to curves along time t. Path-time (PT) diagram, or sometimes called as the *spatio-temporal analysis of motion*, for the example considered is given in Figure 5.5.

The blue half-star at time $t = 0.0s$ in the figure indicates the position of the ego vehicle. In the figure presented position values of the red and the green vehicles are referenced to the ego vehicle. In practical application, these values are based on sensor data, whereas the positions from $t = 0.0s$ on correspond to predicted trajectories. Their calculation is based on the instantaneous velocity at time $t = 0.0s$. Figure 5.5 reflects that the rear axle centerpoint of the green and red vehicle are expected be on the path of the ego vehicle from $t = 4.5s$ and $t = 11.0s$ on, respectively. While the vehicles enter the path, the ego vehicle must preserve some safety distance l_{safety}, as illustrated on Figure 5.6. The safety distance is generally a function of the relative speed among vehicles, but for sense of simplicity, it will be taken as a reasonably high constant value. As a result of the required safety distance, the area occluded by a vehicle cannot be described by a point any more, but will rather be transformed into a line segment. Thus, when the line segment is extruded in time-space, its area on path-time diagram will have the form of a quadrilateral.

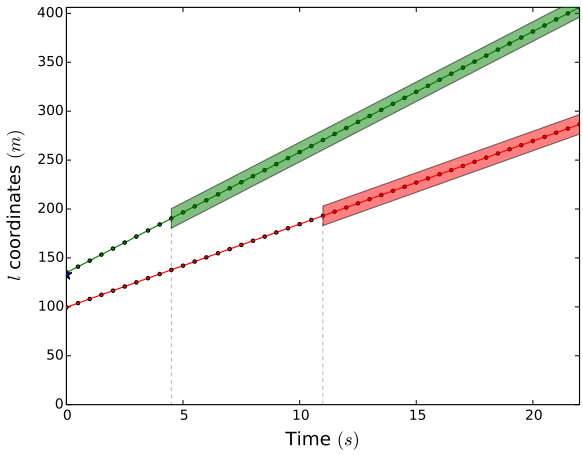


Figure 5.5: Motion of other vehicles plotted on path-time diagram. The dots on the diagram represent sampling instances, and the colored areas represent the restricted l-coordinates in order to maintain some predefined safety distance. They appear once the respective vehicle is on the path of ego vehicle.

Figure 5.6: Region occupied by other vehicles, while maintaining some safety distance l_{safety}, after $t = 11.0$.

The region illustrated on Figure 5.6 represents the l-coordinates, that are restricted in any combinatorial alternative. This restriction arises from the traffic regulations. The combinatorial alternatives will restrict additional regions in l-coordinates. The case for the combination '10' is depicted in Figure 5.7. The extended green and red colored regions are restricted by the combinatorial alternative. The range of l-coordinates, on which the ego vehicle can be located, can be given with l_{feasible}.

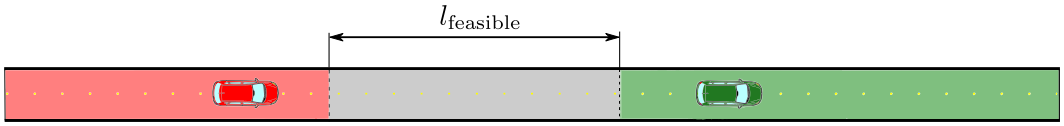
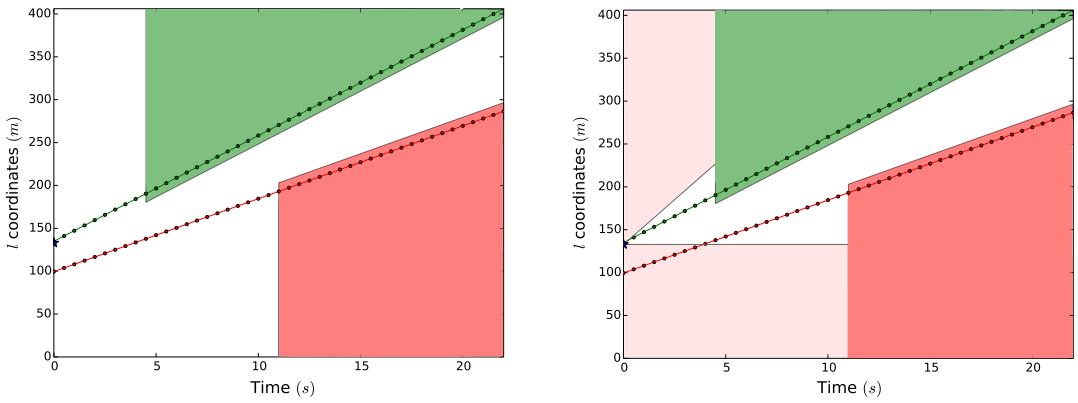


Figure 5.7: Representation of feasible l-coordinates for the combination '10'. The infeasible regions are colored with red and green.

The path-time diagram is very suitable for visualization and evaluation of such combinatorial alternatives. Any sequence with respect to a vehicle can be illustrated by proscribed areas. When these are incorporated into the safety distance violating areas, the resulting quadrilateral will correspond to the *combinational constraint violating area*. An illustration for the combination '10' is given in Figure 5.8a and the diagrams of other combinations are given in Appendix A.3.



(a) Path-time diagram extended to represent combinational constraints.

(b) Path-time diagram with speed constraint reference.

Figure 5.8: Representation of the combination '10' on path-time diagram.

It was previously discussed in this subsection that the feasibility also depends on criteria emerging from the internal constraints of the vehicle. Among those constraints, the speed constraint can easily be represented by a line on the path-time diagram. In this case, lines with lower slopes would satisfy the speed constraint and are therefore reachable by vehicle. The pink area at the upper part of the Figure 5.8b represent motions that would violate the speed constraint. Within the context of this work, backwards driving is prohibited. Hence, at any instance of motion the signed speed must be positive. This can be represented by the pink area at the lower half of the figure. It should be remarked that these regions are only qualitative for the very first timestep. Parallels of those lines have to be taken as reference for any other timestep.

For a combination to be assessed as feasible, it must be able yield a feasible curve from its initial position until the end of the preview horizon. Expressed in other words, there must at least one curve connecting initial position to any position at final time, without intersecting the constrained areas on the path-time diagram and requiring a speed value beyond the feasible range. Apparently, for the combination '10' multiple such curves can be found, where only one of them will correspond to the optimal trajectory. However, if the combination '01' would be regarded, no line can be found. The analysis of whether any such line exist is in fact an analogous problem of procuring an appropriate initialization. Therefore, the feasibility check of a given combination will continue during initialization.

After a combination is found to be feasible, optimal motion for that combination can be obtained by constraining the optimization problem, that was introduced in Chapter 4, with additional constraints. It was mentioned that the ego vehicle can either be heading in front of or behind of a reference vehicle. However, how much in front of, or similarly, how much behind of the reference vehicle is insignificant while defining sequencing. The ego vehicle can have any arbitrary distance to the reference vehicle. Mathematically expressed, such a statement corresponds to an inequality. Considering

the case illustrated in Figure 5.4, the intervehicular distance can be given as:

$$l_{\text{v2v}} = l - l_{\text{ref\_vehicle}} \,. \tag{5.4}$$

In this case, for a given combination the feasible range of ego vehicles l-coordinates can be represented by the inequality

$$b_{\text{sign}}(l_{\text{v2v}} + b_{\text{sign}}\, l_{\text{safety}}) > 0, \tag{5.5}$$

where b_{sign} represents the 'sign' of a binary reference. In case the vehicle is the lead vehicle, the binary index would be equal to '1', and its b_{sign} will also be equal to 1. Where the vehicle is being leaded, then its binary index would be equal to '0', and its b_{sign} will also be equal to -1. Such a formulation can be applied with respect to all of the vehicles in the environment, and thereby the parameter interval of the optimization problem can be constrained for a given feasible combination.

5.2.3. Initialization of the Constrained Problem

The role of initialization on the convergence rate of optimization was explained in Section 4.2. The same discussions apply to kinodynamic planning. Because the path-time diagram represents an overview of the region of feasible motion, it can be used as a basis while performing the initial guess. Colored regions on the diagram can be interpreted as an obstacle and the vehicle's l-coordinate at $t = 0.0s$ can be regarded as the initial position of the autonomous agent. The speed constraint can imposed as restriction subject onto the motion of the agent. Thus, the initialization problem can be regarded as finding a path from the initial position of the agent to an arbitrary position at $t = T$. This corresponds to a trivial path planning problem.

Previous discussions addressed the reasons behind predicting the speed of other vehicles as constant. It was also shown in Subsection 5.2.2 that such an assumption yields linear curves on the path-time diagram. This leads the "obstacles" on path-time diagram to have polygonal shapes. In Subsection 2.2 various alternative methods on path planning were presented. The visibility-graph approach among them, was introduced as a very fast, practical and simple to implement method for the case of polygonal obstacles. On the other hand, the fact that it delivers results that come too close to the boundary of the obstacles was regarded as a deficiency of the method. However, when alternative combinations are searched for, the deficiency evolves into a desirable property. This results from the fact that, moving at the upper and lower bounds prevents the elimination of feasible alternatives. If, for example, the median of upper and lower bound would be used, which very roughly represents the cases in artificial potential field methods and the Voronoi method, then after several timesteps, some corner points would become impossible to reach. An illustration of alternative methods is given on Figure 5.9. The trajectories obtained by Voronoi graph and potential field method in the figure do not satisfy the speed constraint. It is even hard to check which parts of the trajectory do not meet the constraint. In contrast, as visibility graphs are created by *lines of sight*,

the inspection on feasibility of motion can be performed by checking individual lines. Of course, both of these methods may be extended in such a way that they observe the constraints and find paths appropriately. Or as a further option, sampling methods may also be employed. However, all of these alternatives will entangle the initialization and while a very simple method is present, none of them can become prevalent.

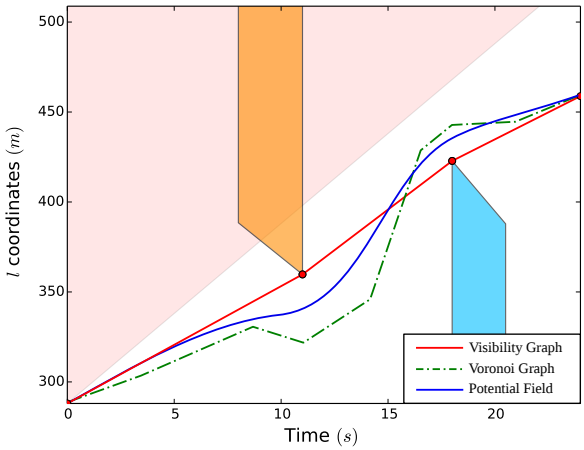


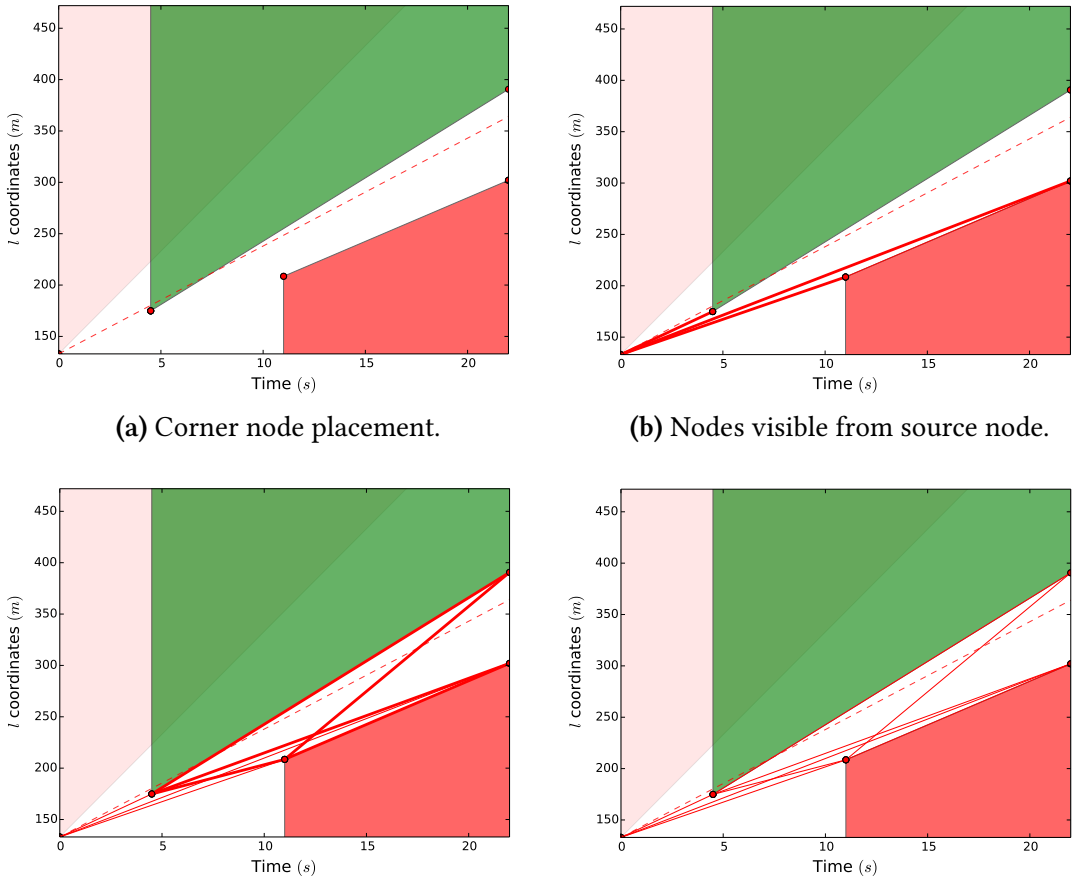
Figure 5.9: Illustration of alternative path-planning methods for initialization. After $t = 11.0s$, in order to reach the tip of the blue obstacle, the paths obtained from Voronoi and potential field methods require a higher speed than allowed. In contrary, the basic visibility graph approach intrinsically overcomes the problem by getting to the obstacles as close as possible.

A visibility graph is created by connecting all of the vertex of the polygons in the environment, that see each other. Each of such lines of sights correspond to edges and once all of them are found, a tree-graph is constructed. Afterwards through the utilization of graph search algorithms, such as breadth-first-search, depth-first search etc. all of the alternative paths are found. Among the alternatives, the path that yields the shortest distance is selected. Connecting all of the vertex causes lots of edges to be found. However, many of these vertex will not satisfy the speed constraint, or even will not be accessible by the robot. Even though they will not make a contribution to the solution, they will slow the path search down. In sparse environments, where the amount of obstacles are limited, this may not be a problem at all. But, if there are also numerous obstacles, these useless vertex may considerably slow the search progress down.

A practical way is to create lines of sight incrementally, while at the same time checking if they hold the speed limit constraint. In this case, starting from the the point where the robot is located, *i.e. source node*, only the nodes that are visible from the source node and obey the constraints are added as an edge. Performing this operation recursively results in a *reduced*[1] visibility diagram and this facilitates the search process. The steps are illustrated in Figure 5.10. The dashed lines in the figure correspond to traveling

[1]Deviating from its formal definition, the term *reduced* is used for the elimination of infeasible edges.

with the desired speed. Of course, this line serves only as a reference. Because of the obstacles present, and due to the geometry of the road such a travel is unlikely to be achievable. Figure 5.10d shows different alternative paths that end up at $t = T$. There is



(a) Corner node placement.

(b) Nodes visible from source node.

(c) Visibility of the nodes that are connected to the source.

(d) Final form of visibility graph.

Figure 5.10: Illustration of steps of visibility graph. The dashed line represents the desired travel speed.

not only a variety of paths, but there are even two distinct final nodes. One of them has a final l-value at about 300m and the other one at about 400m. Among the paths, the one that terminates as close as possible to the desired-terminal node has to be selected. But the selected initial-trajectory must also have a profile that is at closest to the desired profile. This can be checked by summing the hypotenuses of the individual line segments, constituting the motion profile. Furthermore, as any corner point in a visibility graph brings about instantaneous changes in velocity and its derivatives, their number along the path must also be minimized. Each of these considerations with their priority play a unique role during the extraction of the best initial-trajectory. Several tests were performed in order to determine the selection sequence. The tests have proven that, as the primary criteria, the alternative that is at closest to the desired node has to

be selected. If there are still multiple candidates, then the aforementioned approach of summing hypotenuses must be applied. In this case, the found alternative corresponds in general to the best initialization obtainable. Furthermore, the requirement of passing over least number of nodes is also intrinsically satisfied by the condition of similar profiles, *i.e.* by comparing segment lengths.

The figure reflects a further problem while obtaining the best-initialization. Even when the initial-trajectory that mimics the desired travel profile at best is selected, the speed at the last section of the profile will be the same to the speed of one of the obstacles present. Hence, although the vehicle can travel with its desired speed, the approach so far would cause the ego vehicle to drive at the speeds of other vehicles. This resembles the optimized motion profile very poorly. As mentioned previously, the convergence properties depend a lot on the initialization done. Therefore, it must be endeavored for initializing the problem as good as possible. Of course, the analysis done on the path-time diagram disregards geometry of the road and the weighting factors of the individual summands in the cost functional, albeit they have decisive role on the quality of initialization. For example, if the weighting factors are tuned to promote sluggish behavior of the vehicle, then the number and the magnitude of instantaneous changes in the initialized trajectory will determine the quality of initialization. Nevertheless, the discussion about influence of initialization on optimization will be left to Section 7.3, and focus on gaining the best trajectories through the utilization of visibility graphs will be given to. For resolving the mentioned problem, an *improved* visibility graph approach can be proposed.

The improved visibility graph approach is founded on sketching from each of the nodes a line of sight with the desired speed until $t = T$. If it does not hit any obstacle, then the node it terminates is added together with its corresponding edge to alternatives, as illustrated in Figure 5.12a. The added nodes and edges are represented with turquoise color in the figure. The search-tree of the visibility graph is given in Figure 5.12b. The nodes are colored in compliance with the color of the obstacles and are labeled according to a certain rule. The first character, the letter 'O' represents that it is an obstacle. The second character, which is a number, represents arbitrarily chosen obstacle number. The last character represents to which corner of the obstacle the node corresponds to. As illustrated on Figure 5.11, starting with the upper left corner, in clockwise direction, the labels are A, B, C, D. The obtained terminal nodes are labeled with the 'T' as the first character and followed by a random id-number. As indicated in Figure 5.12b, extending the graph with additional nodes and edges results in greater number of alternative paths, making the graph-search process more complex and slower. There is however

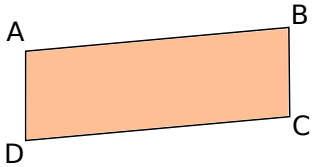
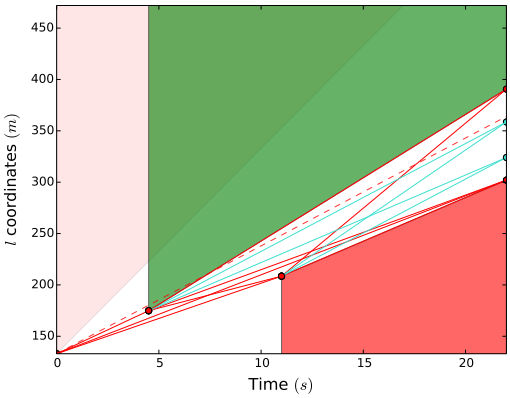
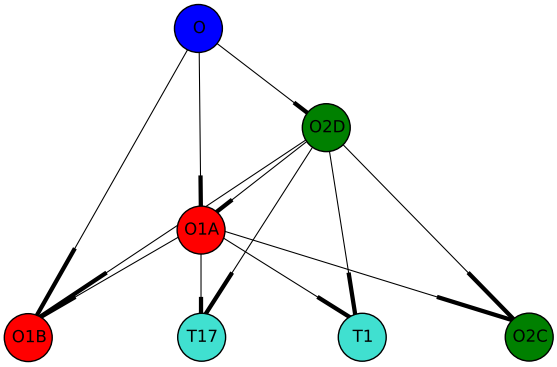


Figure 5.11: Rule followed while labeling an arbitrary obstacle.

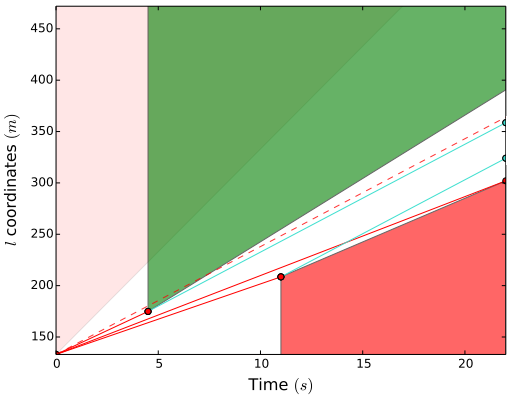
a remedy for this situation. Once a desired-node is visible from a corner node, all other visible nodes from that node will become irrelevant, as they will represent an apparent undesirable choice. Therefore, such edges can be eliminated from the list and a purified visibility graph can be obtained, as illustrated in Figure 5.12c. The corresponding tree-graph is given in Figure 5.12d. It significantly consists of fewer nodes and edges, which enhances the graph-search speed and yields an initialization better than the ordinary visibility graph can yield.



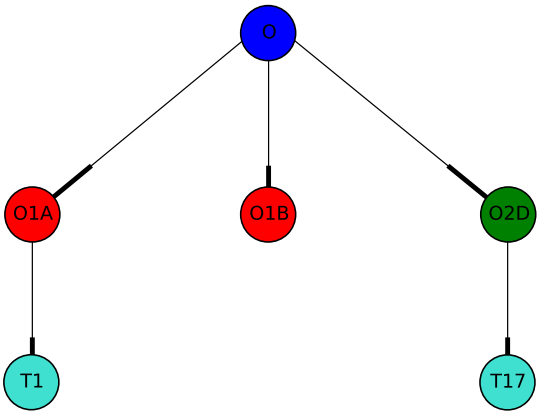
(a) Extending the visibility graph.



(b) Tree-graph of the extended visibility graph.



(c) Purification of visibility graph.



(d) Tree-graph of the purified visibility graph.

Figure 5.12: Steps of improving visibility graph.

Extraction and initialization of all of the feasible combinations can thereby be completed. A flowchart providing an overview is also presented in Figure 5.13. Once the initial trajectories are attained, these will be sent to the solver and the trajectories will be optimized. If there are alternative combinations and if the vehicle has multiple cores, then the optimization for each of the combinations can be run in parallel. In dense urban traffic, where dozens of combinatorial options are present, this would reduce the computation time.

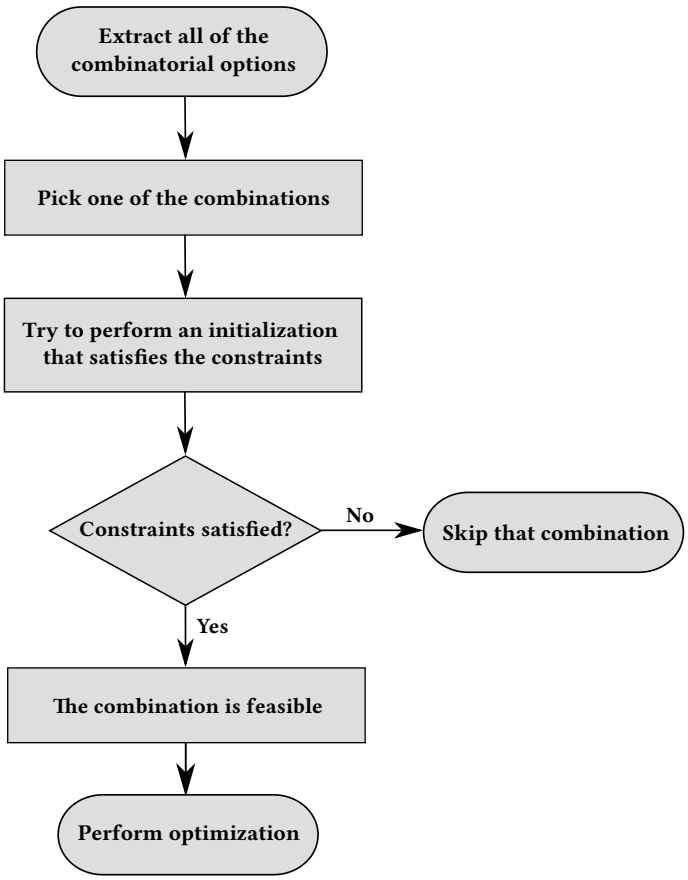


Figure 5.13: Initialization flowchart of kinodynamic planning.

5.3. Path-Velocity Decomposition

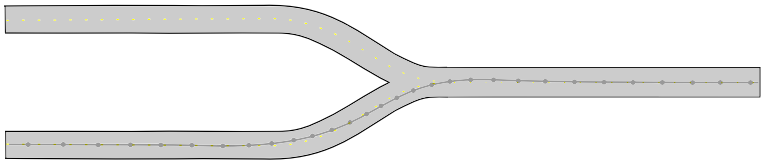
Trajectory planning so far was done along a driving corridor, with utilization of both longitudinal (l) and lateral (d) coordinates. This had introduced two parameters for each node into the optimization problem. The computational burden during optimization typically grows polynomially with the number of parameters. This causes large horizon selection to become intractable whenever the computational power is limited. Especially, if the combinational alternatives are computed sequentially, this would emerge as a serious problem.

A remedy is to utilize *path-velocity decomposition* (PVD). In this approach, path is planned offline and then during execution, in order to avoid potential collisions with kinodynamic obstacles, only the speed is regulated. Thereby, time parameterization of trajectory planning is decoupled from path-planning. If the velocity of the vehicle is calculated on Frenet coordinates, the longitudinal component, *i.e.* the l-coordinates, is adequate for calculating the velocity. In this case, it can be sacrificed from the exploit of lateral coordinates. This corresponds to optimizing only the longitudinal coordinates throughout the path. In this way, each node will introduce a single parameter and

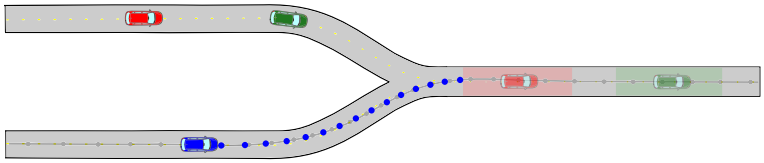
especially at long horizons, this would notably reduce the computational costs.

For obtaining the path on which the velocity will be regulated, the trajectory optimization introduced in Chapter 4 can be used. First, the trajectory optimization is run offline. Then, the resulting path is taken as the reference for l-coordinate computations. Alternatively, the street center can also be used as the reference. However, independent from the selection of the path, the motion of a vehicle obtained via PVD would be a like the motion of a streetcar. As the rails of a streetcar are fixed, only the longitudinal motion is controlled. Escape maneuvers from obstacles will hence be not realizable.

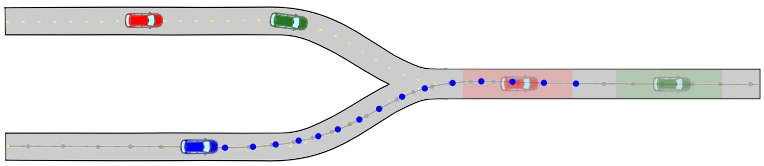
The subsequent steps of PVD for the previously inspected example is shown in Figure 5.14. First, the trajectory is planned offline for an obstacle free environment . Afterwards, in order to maintain smoothness among the trajectory support points, a spline is fitted. Subsequently, during planning in kinodynamic environments, the l-coordinates are optimized for each of the combinations.



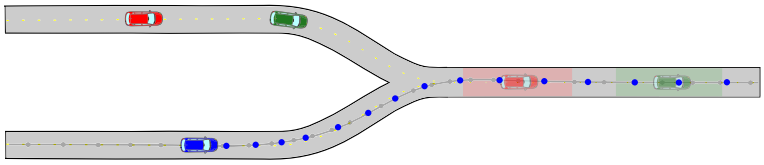
(a) Offline created optimal trajectory.



(b) Velocity planning for the combination ‘00’.



(c) Velocity planning for the combination ‘10’.



(d) Velocity planning for the combination ‘11’.

Figure 5.14: Steps of path-velocity decomposition.

6. Implementation

Discussed considerations so far were devoid of any focus on implementation. During practical application, several crucial problems are encountered. This chapter is devoted to the discussion of implementation related aspects and to resolve them by providing necessary formulations.

The chapter starts with introducing the environment model to which the simulation is built on. Within this section, elements while mapping the real-world, such as the lane information and vehicle positioning, are discussed. These considerations constitute the main subject of this chapter. The chapter continues with presenting the optimization library and the solver used. Then the essentials of the developed simulation program are presented. These comprise the selection of the language, structure of the settings file and the design of the output-GUI. Finally, several further remarks considered during implementation are given.

6.1. Environment Model

A precise environment model is a key element for autonomous vehicle navigation and localization. A good environment model must represent geometrical features of the road, but furthermore must transmit the topological information required. Among various alternatives, one option is to define the environment on OpenStreetMap (OSM) [51], [52]. OSM is a project that aims collaborative editing of geographic data from all over the world. It uses three key elements to model the world in WGS84 coordinate frame: *nodes*, *ways* and *relations*. Nodes correspond to point-objects, ways represent group of nodes, and relations represent a group of any element-type. To any of these elements *keys* and *values* specifying properties about the elements can be tagged. All of the tagging and modeling can performed in JOSM, an OSM editor written in Java [53]. The OSM-project uses xml format for storing the processed data. Hence, it can be parsed in almost any programming language. It should however be noted that, as the geographical data is stored in latitude and longitudes, for further processing in Cartesian coordinates, they must be projected. This can be achieved, for example, via Mercator transformation [54].

OSM serves as a perfect tool for the topological representation of the environment. However, it has deficiencies in representing the geometrical features of the environment. Information about lane-width, from that stemming local geometrical informa-

tion such as curvature etc. , or driving direction are not included. Such information can however be added through properties in form of 'keys' and 'values'.

Within the context of the current work, road geometry is defined with the aid of satellite imagery. Several merge scenarios from real world are selected and right & left bounds of the driving corridor are defined through the 'way' element-type. The created ways are sequentially associated with the utilization of relations. Thereby, driving routes are formed. In between the driving corridor, vehicles are arbitrarily located via tracks. A screenshot of tracks while being edited in JOSM editor is given in Figure 6.1. Such tracks are added in form of ways, where each node resembles a timestamp. The representation style is implemented without a profound concentration on extendability. A comprehensive, in practice applied approach for lanes and environment modeling is introduced as *Lanelets* and is accompanied by the in paper presented software libLanelet [51].

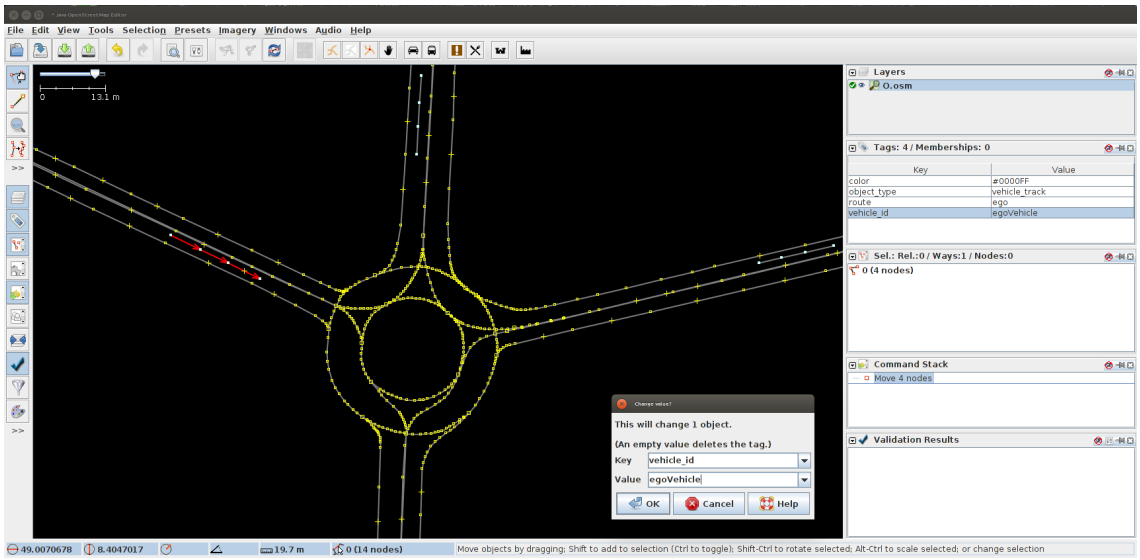
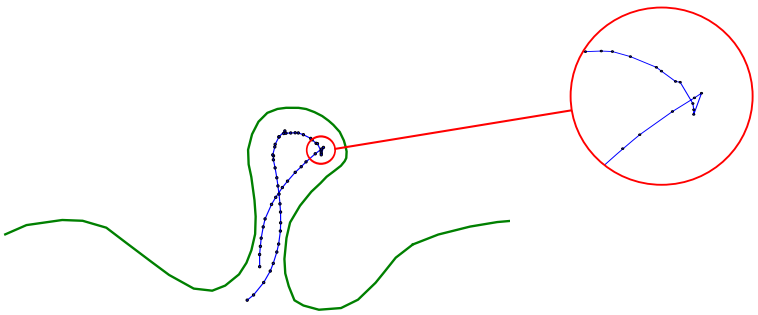


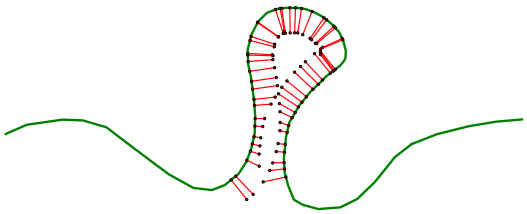
Figure 6.1: Screenshot of JOSM while modifying tracks.

Up to this chapter, the advantages of using the road centerline was several times partly mentioned. Here it will be discussed in a broader perspective. The foremost issue to indicate is that the centerlines can be built up offline, whereas all of the gains that will be mentioned originate from this property.

As discussed in Page 32, the lateral offset calculation inside cost functional will be calculated by a single arithmetic operation, and therefore the computational costs will be at least the half, in comparison with checking lateral distance from both the right and the left corridor-bound. In the latter case, the signed-distance from both boundaries must be retrieved and subsequently be added. As the cost function constitutes the core of the trajectory generation and is likely to be called multiple times for a single instance, this would notably reduce the computation time.



(a) Backwards mapping by specifying some lateral distance from the reference curve. Depending on the value of lateral distance and curvature, a node can be mapped behind a node that has a smaller longitudinal value. This pathological case is magnified.



(b) Forwards mapping done for the blue curve on Figure 6.2a. The nodes are mapped to the closest curve. At the narrow pass, high values of lateral distance caused mapping to the undesired curve. Therefore, the lateral coordinate values of the points in the narrow pass are different from the lateral distance values of the blue curve.

Figure 6.2: Pathological cases that can occur during mapping.

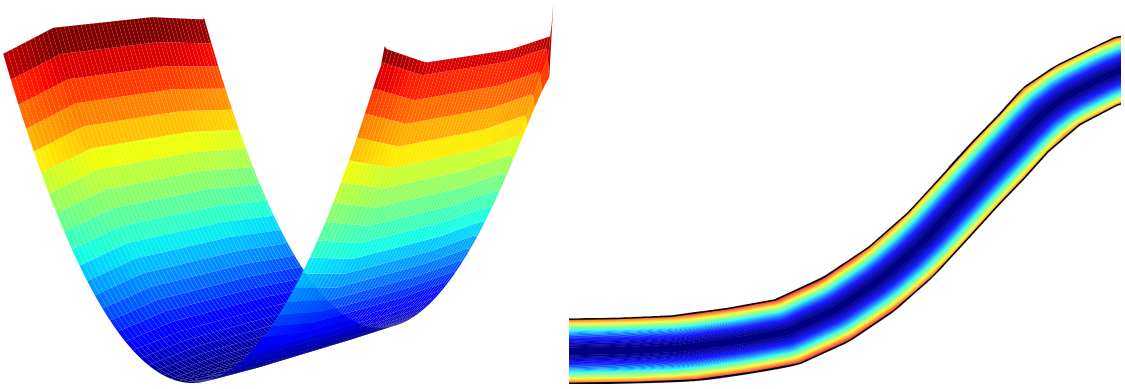
The Figure 4.6 demonstrated that at the center of curves the Frenet coordinates and the Cartesian coordinates deviate from each other at least. As illustrated in Figure 6.2, if the lateral distance of a node to the reference curve is relatively big, then some pathological cases occur. To delimit such undesirable situations, and their misleading results, the reference curve must be in equal distance to the both bounds of the driving corridor. This implies that the most accurate results are obtained through selection of road-centerline as the reference curve. If such cases occur during the centerline computation, they can be recognized and treated by resorting the nodes. The resorting process can be done by arranging the nodes according to their Euclidean distance. Such a treatment is also included in the implementation. However, it should be noted that this has the potential to bring about zig-zagging of the centerline. On the contrary, if the trajectory computation would be based on right&left bounds, then it would even be almost impossible to detect such pathological cases.

Another advantage emerges from the need to predict the motion of other vehicles. If the road-centerline on which they are traveling is known in advance, just by projecting the vehicle's speed along the centerline, their predicted trajectory can be attained.

A further reason arises not only as an alternative but rather as an indispensable need. The calculation of time-to-intersection or time-to-merge with an another vehicle re-

quires the path length from that vehicle's current position until the intersection point to be known. Furthermore, the precision of the path length has influence on the whole computation. An accurate way to measure such lengths is to exploit centerlines, with the underlying reason being similar to the reason behind the using centerlines as reference-curves.

The benefits of creating a centerline and choosing it as the reference are clarified. How the centerline are generated will now be discussed. In fact, centerlines are generated by merely calculating a point that lies to the right & left bounds at the same distance. This can be formulated as an optimization problem, where the objective is to minimize a cost function that consists of quadrature of the sum of lateral signed distances from the both bounds of the corridor ($\mathcal{J} = (d_{\text{right}} + d_{\text{left}})^2$). In this case, the profile of the cost function will be as illustrated in Figure 6.3a , where the minimum will reflect the d-coordinate of a given l-coordinate. If the whole driving corridor would be modeled alike, the cost profile for a trivial path would be as illustrated in Figure 6.3b. The centerline can computed along l-coordinates with finite steps. Due to the quadratic structure of the cost profile, a SQP-based optimization algorithm would be a promising solver. In this case, the need for an initial guess could be perfectly met by introducing d value of the previous l-coordinate (l_{i-1}).



(a) Cross section of the quadratic cost profile.

(b) Top view of the driving corridor.

Figure 6.3: Contour plot of the centerline cost function for an arbitrary driving corridor.

The selected step length for the centerline computation must be small enough to describe sharp turns with a reasonable smoothness. This corresponds to a sampling length of 5-10 cm in practical application. At straight roads, however, a sampling rate of 2-3m would usually be an appropriate selection. Hence, sampling the whole road geometry with a unique interval length is not plausible preference and the sampling-length must be selected according to the curvature of the road geometry. Therefore, a curvature-adaptive step length selection is required. An adaptive sampling method was implemented into the simulation program. Its pseudo-code is given in Appendix A.2. An illustration of the resulting curvature-adaptive sampling for a roundabout is given in Figure 6.5. Notice the increase of support nodes at curves.

It was mentioned that the centerlines serve as backbone behind the time to intersection calculations. Within this respect, all of the roads on which the vehicles are traveling on must be defined relative to the road the ego vehicle is traveling on. That requires all of the roads defined to consist of three road-segments, as presented in Figure 6.4: *before intersection*, *intersection*, and *after intersection*. These segment lengths can be obtained by matching roads to each other and then checking whether the lateral distance between them is below some threshold. The check-points that have a distance to the matched curve lower than the threshold indicate intersection. If the distance exceeds the threshold after an intersection, the segment from the last intersection check-point on will constitute the 'after intersection' part.

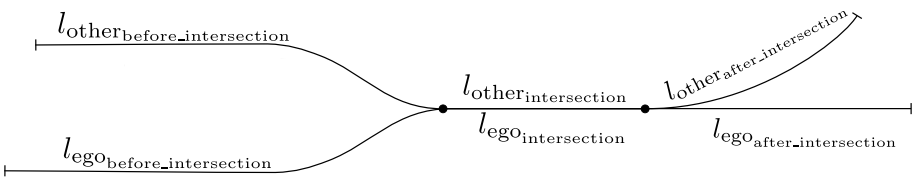


Figure 6.4: Three segments of a road.

The lengths of the segments play an essential role on motion planning of ego vehicle. Not only for returning information about how much a vehicle has to travel until the intersection, but also for determining whether a vehicle is located somewhere on the path of ego vehicle. The check whether a vehicle is on the path of ego vehicle can performed by firstly checking it's l-coordinates, *i.e.* whether it is some neighborhood ϵ of $l_{\text{intersection}}$. If it is inside ($|l - l_{\text{intersection}}| \leq \epsilon$), then the driving corridor check introduced in Chapter 4 on page 36 can be applied. The reason behind twofold check is illustrated in Figure 6.5 below. As shown, in some cases, even when the roads practically intersect, the intersection starts after several meters. Increasing the threshold may be proposed as a remedy but this would sacrifice the precision during departure, at the start of 'after intersection' segment. Hence, the two-stage check is the felicitous solution.

6.2. The Solver: NLopt and SLSQP

For solving nonlinear problems with a SQP method, numerous optimization toolboxes, including open-source variants, such as pyOpt, Scipy, and NLopt, are available [55], [56], [57]. Among them, the NLopt was chosen as the solver for the trajectory optimization problem.

NLopt is written in C programming language by Steven Johnson at MIT and entails numerous algorithms that are callable from C, C++, Fortran, MATLAB, GNU Octave, Python and more with a common interface. The SQP algorithm implemented in NLopt is based on the Sequential Least Squares Quadratic Programming (SLSQP) algorithm by Dieter Kraft [58]. The original algorithm was written in Fortran, but the version in

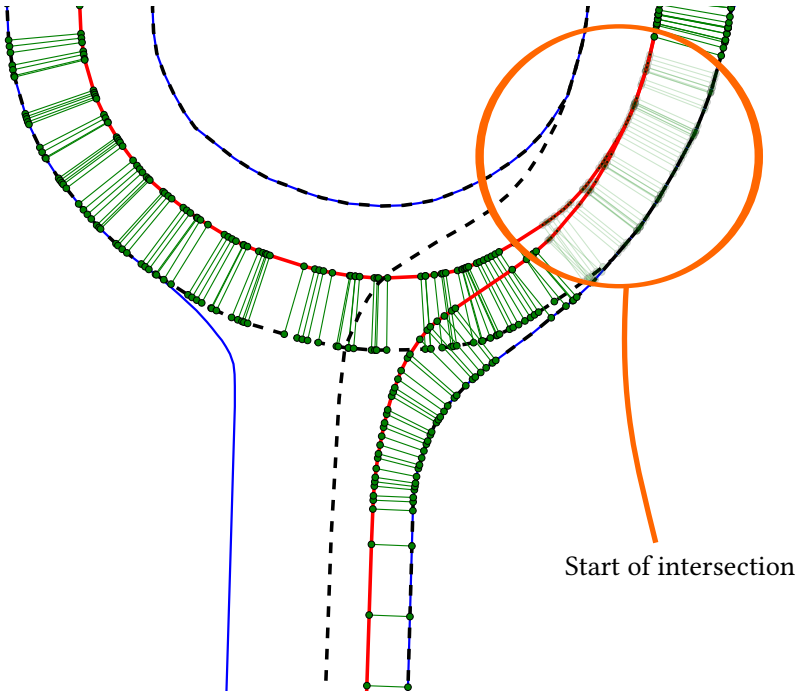


Figure 6.5: Generation of centerline for a roundabout. The centerlines are indicated with red color and the road boundaries with dashed black lines. The green points on the red lines are the support nodes and the green normal lines indicate to which point on the reference curve they are mapped to. Start point of intersection is indicated with an orange ellipse. It should be remarked that, in practical terms, the intersection begins prior to the start point.

NLopt is reimplemented in C and has been enhanced towards increasing efficiency and robustness. The other alternative, pyOpt, contains several SQP algorithms, including the SLSQP. So does also Scipy. However, they both, as result of being limited to Python, are unlikely to be supported and documented as well as NLopt.

The SLSQP algorithm uses the ordinary BFGS update, which is a dense matrix method, for approximating the Hessian [58]. The step size α is determined by the penalty function proposed by Han [59]:

$$\phi(\mathbf{x}_k, \boldsymbol{\rho}_k) = f(\mathbf{x}_k) + \sum_{i=1}^{p} \rho_{k_i} \left| i\,\mathbf{x}_k \right| - \sum_{i=1}^{m+p} \rho_{k_i} \min\left(0,\ c_i(\mathbf{x}_k)\right), \tag{6.1}$$

where ρ_{k_i} is the penalty term of the i^{th} constraint c_i, and can iteratively be computed as suggested by Powell [60]:

$$\rho_{k_i} = \max\left(\frac{1}{2}\left(\rho_{k_{i-1}} + \left|\lambda_{k_i}\right|\right),\ \left|\lambda_{k_i}\right|\right). \tag{6.2}$$

The λ_{k_i} represents the Lagrange multiplier of the constraint c_i, whereas the p and m represent the number of equality and inequality constraints respectively. A step size $\boldsymbol{\alpha}$

that reduces the value of the merit function γ will be chosen:

$$\gamma(\boldsymbol{\alpha}) = \phi(\mathbf{x}_k + \boldsymbol{\alpha}\mathbf{p}_k). \tag{6.3}$$

The quadratic problem is then solved using an equivalent linear least squares problem, whose solution was proposed by Lawson and Hanson [61].

6.3. Overview of the Simulation Program

The simulation program is written in Python. In contrast to C++, Python offers ease of implementation and good readability resulting from its high-level structure. In order to start the analysis, a settings file in form presented in Appendix A.1 is entered.

After the simulation is started, OSM data specified in the settings file is parsed via xml Python binder lxml. Next, the check for the existence of a file that contains road topology and geometry data is performed. If such a file is not found, centerlines for all of the roads defined inside the OSM file are computed and matched with each other, and are subsequently stored in a pickle file for later simulations.

Once the road data is available, together with specifications about temporary objects introduced in OSM file, such as information on obstacles and ego vehicle, the environment-object is constituted. At the next step of the optimization process, information about traffic participants are extracted from the environment. Based on the introduced tracks and timestep step-width defined in settings file, a spline is fitted for the tracks of the vehicles. Depending on whether the step-width set is greater or smaller than the timestamps of the tracks, interpolation or extrapolation is performed. The purpose of this process is to set the vehicle timestamps and the analysis-times on a common basis. The initial velocity information of other vehicles are extracted from those recreated timestamps. For the ego vehicle, besides its speed, also acceleration and jerk values are calculated. Notice that, obtaining such informations for the last step prior to the begin of trajectory generation entails introduction of at least three timestamps for the ego vehicle, and two timetamps for other vehicles within the OSM file. This is an expected result arising from the use of finite differences, as introduced in Section 4.1.3.

The necessary data is then processed using Numpy [62], whereas splines were fitted using the interpolation module of Scipy [56]. It should however be noted that all of the data used was first normalized before the spline was fitted. The retrieved data from spline was later multiplied with the normalization factor. Overlooking the normalization process leads to poor fits.

The gradients required for the optimization algorithm are computed numerically, using the module Numdifftools [63]. Finally, the optimized results of combinations are stored in JSON format for portability. However, all combinations with their optimization details are also stored in a pickle file further inspection.

For visualizing computed trajectories, the obtained motion profile is animated on a GUI and the crucial parameters affecting the computation are simultaneously printed to the console. A screenshot of the GUI and the form of the printed outputs is given in Figure 7.1 and 6.6 respectively. On the left half of the GUI, vehicle motion is plotted onto the imagery obtained from OSM. The bar below allows to focus onto the vehicle. On the right half, the motion diagram is simulated trajectory is plotted. This consists of plots of x- and y-components of velocity, acceleration and jerk. It was shown in Subsection 4.1.2 that the constraints, in their simplest form, bound the quadrature of the values. If these scalar values, such as the speed, is desired to be viewed, the magnitude button located at the right-lower corner of the GUI can be clicked. This will activate plotting their magnitude. The animation can also be paused and a screenshot at any instance can be taken with the aid of the other two buttons next to the magnitude button. Also, in cases where the plots of x- and y-components intersect, in order to view any of them in single, they can be activated or deactivated by clicking onto their corresponding legend. The

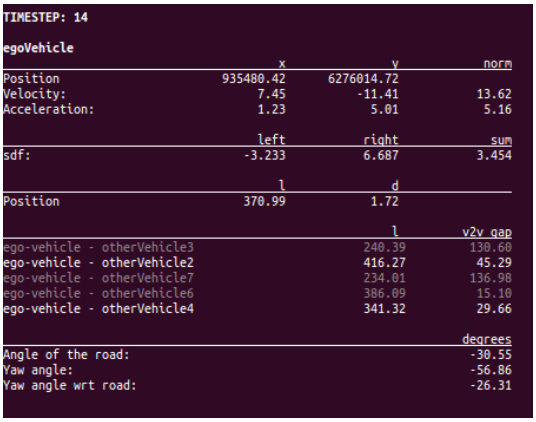


Figure 6.6: Terminal outputs accompanied with the output-animation.

terminal outputs print the details of motion, *i.e.* position, velocity, acceleration values, the signed distance to the right&left bounds of the driving corridor, distance to all of the vehicles present, and also the instantaneous yaw angle, road angle and road-relative yaw angle.

In order to inspect and understand the optimization process, trace of all of the evaluated parameters are stored. These parameters are later visualized on a GUI through an animation. The corresponding GUI is depicted in Figure 7.9. The buttons in the GUI are similarly arranged. Additionally, through the use of 'show optimum value' button, it can be switched between the currently tested parameters, and to the optimum reached so far.

6.4. Further Remarks on Implementation

There are further issues that do not fit into the sections given so far. Therefore, such issues and considerations will be discussed in this section.

First of all, as mentioned in Chapter 1, there are some assumptions made during the simulation. The most fundamental one is that, the position of other vehicles are assumed to be perfectly known. In other words sensor uncertainty is neglected during the simulation. However, if the uncertainty is to be included, the obstacles in path-time diagram can be grown according to the degree of uncertainty. A more realistic approach will be presented in Chapter 8, while mentioning future work. A further condition on the analysis was more like a pursued requirement rather than a simplification. The planning is done in such a way that, the motion of the ego vehicle does not have any influence on the trajectory of other vehicles. Explicitly stated, planned trajectories do not necessitate acceleration or deceleration of other vehicles. Thereby, the motion of ego vehicle will theoretically not interrupt the behavior of other vehicles and the ego vehicle will glide along the traffic flow.

The trajectory and prediction formulations were obtained for a single instance, *i.e.* planning was done quasi-statically. However, if the refresh rate of the computation is above some level, the planner will be able to deal with the dynamic obstacles present in the environment. Otherwise the ego vehicle might become passive. In practical application refresh rates above 5Hz yield desirable results. The required refresh rate is however not only determined by the dynamicity of the environment. If the sensor uncertainties are high, the refresh rate must be increased further for collision-free planning.

The presented implementation aims the study of factors affecting the trajectory optimization and combinatorial variations. In this respect, tolerance value is selected as the termination criteria. The chosen tolerance was the relative tolerance of the function value. The reason behind the selection of relative tolerance is that, unlike absolute tolerance, it exploits the whole interval of machine precision. Absolute tolerance in contrast can only deliver half of the machine precision reliably. In fact, using absolute parameter tolerance would yield better results. Such a termination criteria is independent of the length of the curves on the road, which inherently increases the cost of planned trajectories. Actually, in a real-time implementation, as the planner has a bounded time to return a result, a termination criteria based on tolerance value becomes irrelevant. The only option in such a case is to set a time limit on the optimization process.

It should be emphasized that the effective planning horizon value is obtained by subtracting the computation time from the set horizon value. For fail-safe design, in case no result is found within a specified time, the vehicle must be programmed to decelerate and eventually to stop. As an alternative, in such cases the planning horizon can be reduced and trajectories for shorter horizons can be obtained. Of course, if this alternative is selected, then whenever the horizon value falls below some critical-level emergency braking must be activated.

As the simulator program is implemented for checking feasibility of the presented approaches, once the analysis is started, visibility graphs, search-tree diagrams are automatically stored for each combination and for each time instant. Of course, if any obstacle will cross the road of ego vehicle, then for initialization the visibility graph approach will not be called.

During the implementation of the simulator program, time of computation was not a primary design criteria. It is also a reason behind the selection of Python as programming language, the use of numerical derivatives, and an optimization algorithm that relies on Hessian approximation. However, considerations reflecting real time application has always been given throughout the thesis. Besides remarks given so far, if analytical gradient and Hessian information would be provided, real time implementations could be achieved [34].

Previously, it was mentioned multiple times that a transformation between Cartesian coordinates and Frenet coordinates is done. For the transformation, a pseudo-distance function is used. The details about its implementation is given in [51] and [12], hence will not be repeated here.

7. Results and Evaluation

Outcomes of the introduced formulation are presented and evaluated in this chapter. The chapter starts with presenting several scenarios encountered in everyday traffic. Optimized motion profiles are given for all of the feasible combinations of the inspected scenarios. Once the analysis of complete planning is complete, one of those scenarios is selected and based on that, a comparison for the trajectory qualities is made between complete planning and decoupled planning. The third section of the chapter investigates the effect of initialization on the result and the convergence speed. While recapitulating local optimization, it was emphasized that the initial guess must be in the vicinity of the solution, otherwise the convergence might fall in danger. Hence, keeping initialization close enough to the optimal result has been a matter of substantial endeavor. However, the measure of '*close enough*' was left unmentioned. In order to define this measure and to reveal its influence, a problem is initialized once very close and then quite away from the solution. The investigation is done for both complete and decoupled planning. As the optimization algorithm plays a decisive role on the obtained trajectory, the optimization procedure of the selected algorithm is also examined within this chapter. Subsequently, with some further remarks the chapter is concluded.

7.1. Inspected Situations and Results

A variety of different combinatorial problems occur at everyday traffic. Certainly, the motion planner must be able to yield desirable results independent from road geometry and the number of obstacles. For that purpose, the representative example used in Chapter 5 is replaced by several real world scenarios with various road geometries. Some of the selected scenarios are given in Figure 7.1.

The initial tests for the trajectory optimization were done in absence of other vehicles. This removed the obligation to handle kinodynamic obstacles and therefrom arising complexities. The primary goal was to test the simulation program and to tune the weighting factors. Weighting factors can be best tuned in curved roads. Such roads pose conflicting goals on the motion planner. The frequent curves cause deterioration of driving comfort, where the planner can only ameliorate it in expense of deviating from its desired trajectory.

A very winding road appropriate for such an analysis is the Lombard Street, located in San Francisco, USA. A trajectory planned on Lombard Street is given in Figure 7.1a. The

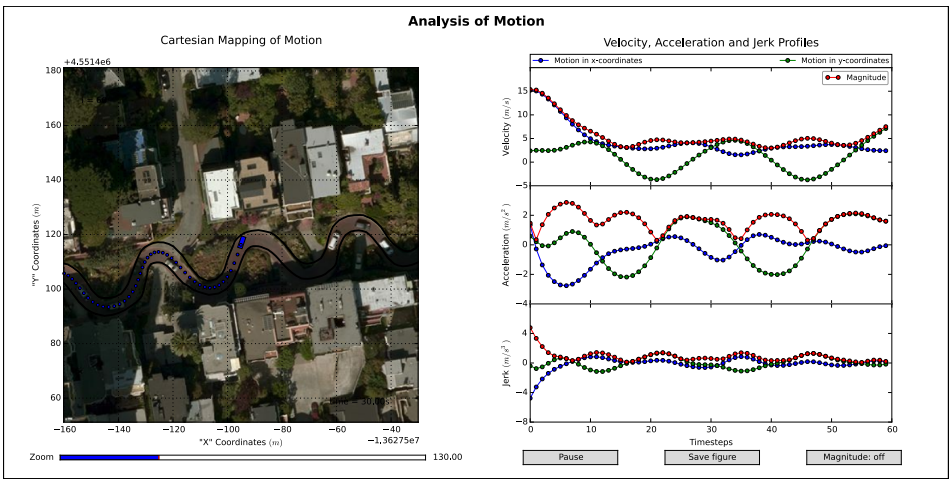
Cartesian mapping of the optimized motion is given on the left hand side, whereas its profile is given on the right hand side of the figure. The velocity, acceleration and jerk profiles are very smooth and are in compliance with the geometry of the road. As the samples were taken at equidistant timesteps, the frequency of the points on Cartesian mapping represent the speed of the vehicle. It should be underlined that increasing frequency of those points indicate a decrease in the speed. This represents the trade-off made for reducing the cornering forces in against increasing ride-comfort. The vehicle on the figure generally tends to follow the center of the street. It is a consequence of the impact of weighting factors forcing the vehicle to remain on the centerline (w_{track}). If for example, w_{track} would be reduced, or if the weight of traveling at set-speed w_{velocity} would be increased, then a broader range of lane width would be utilized.

A critical issue should further be mentioned. Towards the end of the planning horizon the planner returns an implausible trajectory. Although this may seem an error, it is an expected situation. At the last timesteps the driving corridor does not practically bound the vehicle motion. Hence, the vehicle starts to accelerate, unaware of the fact that relying on those support points will violate the driving corridor constraint and will be subject to major changes after the vehicle has moved a bit. The only remedy is to cross several points towards the end out.

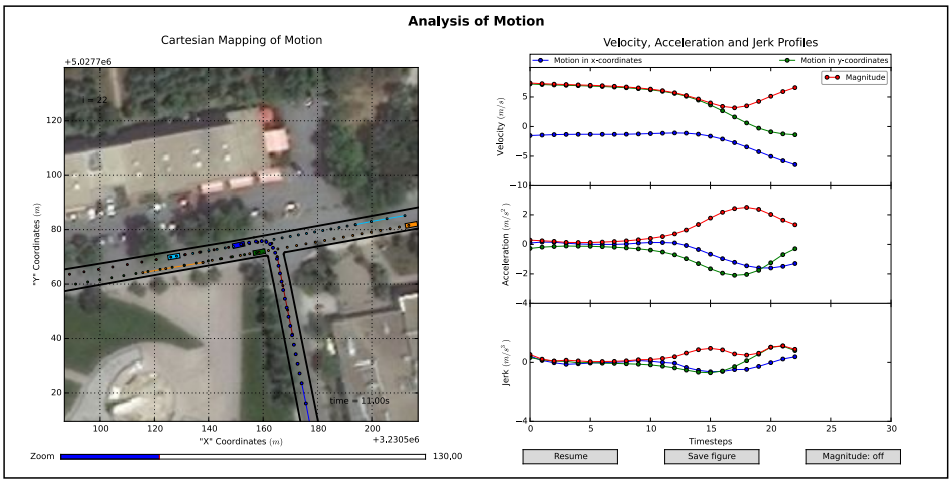
Another difficult problem in autonomous vehicle trajectory planning is posed by uncontrolled t-junctions. A vehicle at a t-junction has not only to observe the vehicles on the route it will merge to, but also the oncoming traffic while crossing the intersection. An example of an uncontrolled intersection from the Istanbul Technical University campus, Istanbul, Turkey is given in Figure 7.1b. In this scenario, the ego vehicle has to turn left, whereas a vehicle (cyan colored) on that lane and two vehicles (orange and green colored) on the opposite direction are also proceeding to the intersection. Moreover, the motion of the ego vehicle is further restricted by another vehicle (dark red colored) heading on the same route as the ego vehicle. The instance given in the figure corresponds to the combination of yielding to cyan and orange vehicles, but moving before the green vehicle. Notice that the vehicle slightly decelerates while approaching the intersection.

The path-time diagram of the scenario is given in Figure 7.2a. On the diagram two characteristic features are clearly visible. First, from the first instance on, the position of the dark red vehicle is grown by the required safety distance. This implies that the vehicle shares the same route with the ego vehicle. The other distinct feature is seen from the inclination of the motion lines. Two of the lines, the green and the orange one have a decreasing inclination. This indicates that these vehicles are oncoming and hence are traveling on the opposite direction.

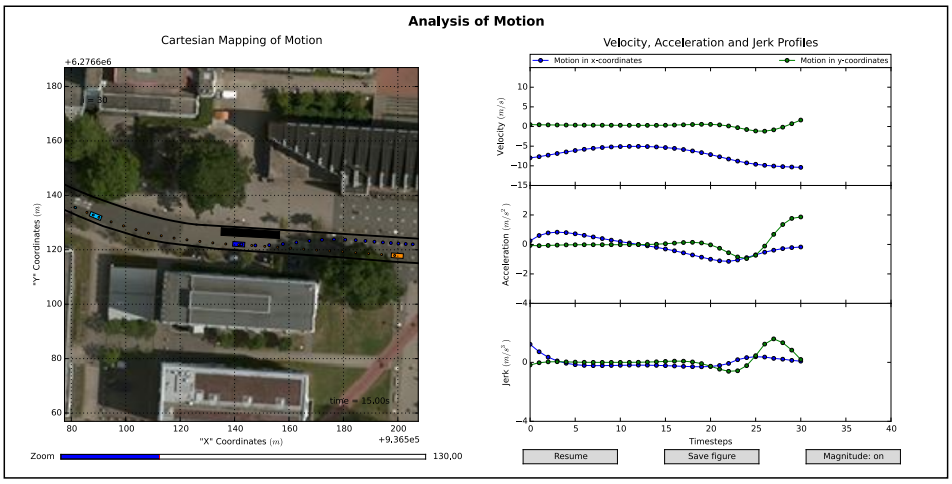
Optimized trajectories of alternative combinations are given in Figure 7.2b. The in Figure 7.1b treated combination corresponds to the curve in between the others. On the curve of that combination, the bending prior to the green quadrilateral is a consequence of the deceleration done before maneuvering. The combination of driving ahead of the



(a) Lombard Street, San Francisco, USA. Hairpin turns allow tuning of weight factors.



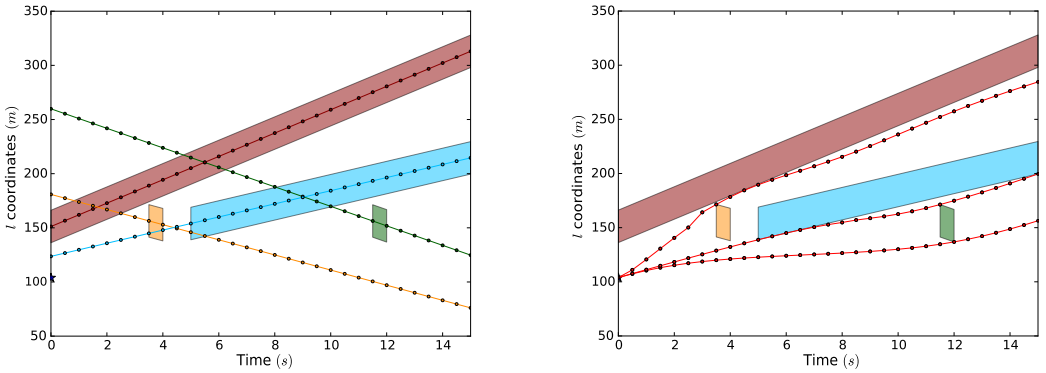
(b) A t-junction at the Istanbul Technical University campus, Istanbul, TR.



(c) A narrow pass in presence of two upcoming vehicles at the KIT campus, Karlsruhe, DE.

Figure 7.1: Several challenging scenarios encountered in every-day traffic.

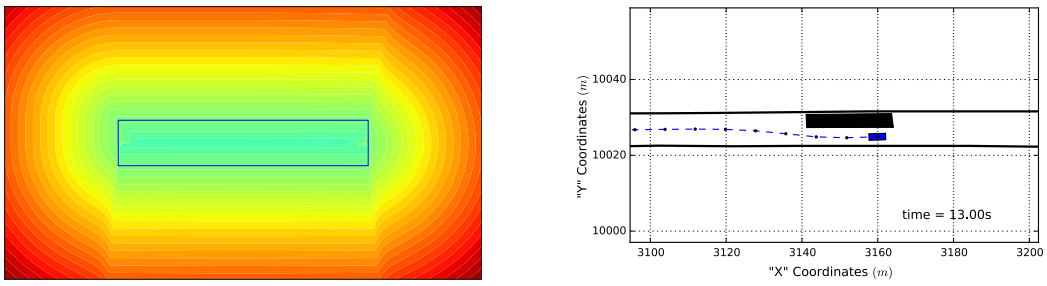
orange vehicle is not very smooth. Especially the node just before the area of the orange vehicle is likely to be an outlier and therefore it indicates that further optimization could yield better results. The question why the motion of this combination is not as smooth as the others will be discussed on Page 73. The computation steps of the profile in Figure 7.1b is given in Appendix A.3 in detail.



(a) Motion of other vehicles on PT diagram. **(b)** Alternative combinations on PT diagram.

Figure 7.2: Spatiotemporal analysis of the t-junction considered.

A more complex problem for autonomous vehicle motion planning occur when the vehicle is allowed to overtake. In this case an additional possible combination arises for every vehicle traveling on the same direction as the ego vehicle. However, modeling such a case is a more arduous task than the intersection scenarios. Nevertheless, it is striven for to resolve such problems and the target is broken down into subproblems. The first step is to model a vehicle as a static obstacle, occupying some part of a road. In such a case, as illustrated in Figure 7.3a, the ego vehicle has to depart from the road centerline and has thereby to escape from the obstacle. By modeling a vehicle as a closed contour and exploiting signed distance, this can successfully be realized. The contour plot of a quadrilateral obstacle is given in Figure 7.3b. The sum of signed distances from the boundaries of the obstacle are positive outside and negative inside the obstacle.



(a) Signed distance function of a closed contour. **(b)** Ego vehicle escaping from static obstacle.

Figure 7.3: Manuevering in presence of a static obstacle.

The next step was to model the road as narrow pass and to focus on maintaining the safe-ride. In fact this does not have any difference than the scenario discussed for a

t-junction. The constraints developed for oncoming vehicles in that case can be applied here in the same way. For testing such a situation, a scenario from the campus of the Karlsruhe Institute of Technology, Karlsruhe, Germany was selected. The optimized trajectory for this scenario is given in Figure 7.1c. A black depicted obstacle, which in fact was indented to represent the KIT shuttle bus, is parked on the route of ego vehicle. From the other direction of the street two vehicles, colored in orange and cyan, are traveling. In the presented figure, among other combinatorial options, the ego vehicle passes the junction after the orange vehicle and before the cyan vehicle. A very smooth trajectory is obtained for such a situation.

The next step on the implementation of overtaking is to model the road such that at any instance the ego vehicle can cross the oncoming lane. Afterwards, by combining the presented escape technique and the formulation used for the narrow pass, an overtaking can be realized. It should however be mentioned that, in this case the analysis of possible combinations, which in fact founds the core of combinatorial reasoning, develops into a very complex problem that is unlikely to be examinable on a two dimensional path-time diagram. Due to time restrictions, such scenarios could not be studied further. However, even resolving the narrow pass problem has been a significant progress for autonomous vehicle motion planning. The discussions introduced will surely serve as a foundation for future studies on overtaking.

How the costs of combinatorial options evolve throughout the time is an important issue. If there is incoherence between the best combinations, *i.e.* if the best combination alter at every planning instant, the planner may undergo instability. A test for evolution of the combinatorial costs was performed for the scenario at the KIT campus. As illustrated on Figure 7.1c, there are two oncoming vehicles. These result in three alternative combinations of passing the corridor before both of them, before one of them, and after all of them. The Figure 7.1c gave the motion profile for the combination 10 at $t = 75.5s$. Costs of alternative combinations from that timestamp on are given in Table 7.1, and the path-time diagram of alternative combinations at several timesteps is given in Appendix A.3.

Combination	Timestamp								
	75.5	76.0	76.5	77.0	77.5	78.0	78.5	79.0	79.5
'11'	46.46	60.67	101.75	155.10	–	–	–	–	–
'10'	17.34	17.32	17.12	16.93	16.69	16.38	16.00	15.56	15.04
'00'	41.78	48.58	49.49	49.96	50.16	50.18	50.16	50.50	51.28

Table 7.1: Cost-evolution of feasible combinations throughout the time for the situation considered in Figure 7.1c.

As presented in the table, for timestamp $t = 75.5s$ the most favorable combination is the '10'. Hence, the planner will select that combination and drive based on its reference

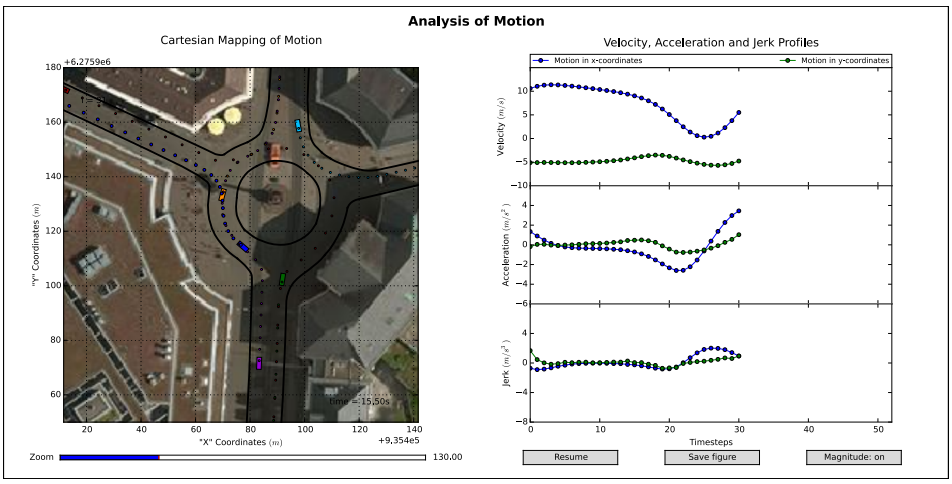
trajectory, until a new trajectory is available. At $t = 76.0s$ a new trajectory is planned. The combination 10 has still the smallest cost value, whereas the costs of other combinations have increased. The costs of other options were expected to increase, however, the cost of the picked combination must not necessarily decrease. Its change depends on the road geometry at the end of the planned horizon. When change in all of the options are compared, the cost of traversing the narrow pass before both of the vehicles has increased significantly more than the cost of driving after. This can be attributed to two basic reasons. First, as the ego vehicle and the orange vehicle are driving in opposing directions, the feasible interval (l_{feasible}) for passing through the narrow pass has decreased considerably. Hence, the ego vehicle needs to hit the gas in order to still be able to drive by the static obstacle before the orange occludes the passage. Such an acceleration naturally results in an increased cost. Second, in the case of driving after both of them, the motion of other vehicles may not be restricting the motion of ego vehicle very much. So, the ego vehicle can apply that combination by just braking slightly. As presented in the table, the discussed trend remains the same along the time. After some time however, the first combination becomes infeasible. The internal limits of the vehicle do not allow after $t = 77.0s$ the ego vehicle to apply the combination '11'.

Roundabouts are another critical intersection scenario encountered in everyday traffic. Several roads from various directions intersect at an ordinary roundabout. The autonomous vehicle has to consider all of the vehicles, that are approaching from different roads and heading to different destinations. So, the situation is quite more entangled than the ones considered so far. Until now, the vehicles remained on the same road, but now they can depart from the route of ego vehicle. This enables the feasibility of additional combinations.

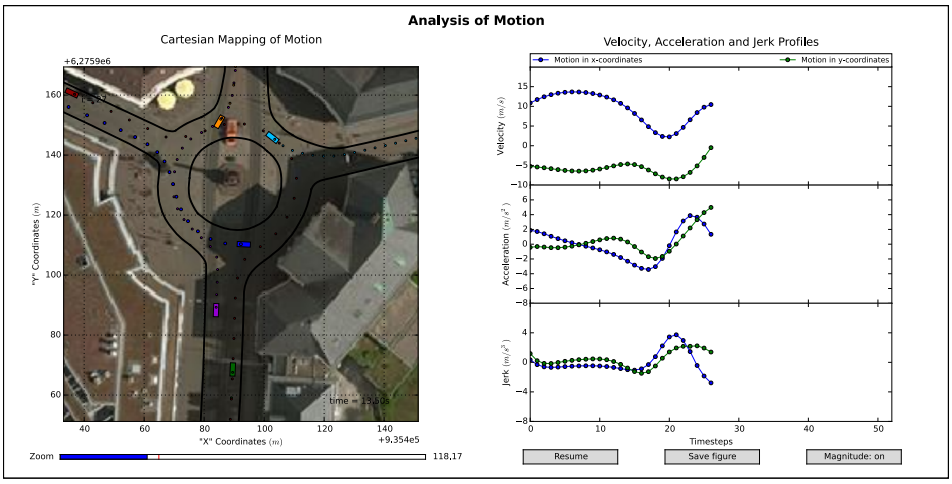
The roundabout scenario selected for simulation is the Rondellplatz in Karlsruhe, Germany. Five alternative combinations are given in Figure 7.4. The analysis was performed with a desired travel speed of $10m/s$. In this case, the expected motion to be picked up is given in Figure 7.4a. The vehicle travels until the intersection at a slightly higher speed than the desired and performs its motion without requiring high lateral offset values. In this case, the vehicle travels before the orange vehicle, but after the purple, green, cyan and the partly visible dark red vehicles.

An alternative combination is to accelerate slightly and enter the intersection before the green vehicle. This combination is illustrated on Figure 7.4b. The increase in speed inside the roundabout has also brought about the increase in the acceleration and the jerk values while cornering. Whenever a comfortable ride is intended, these values will lead to an increase in the cost of this combination.

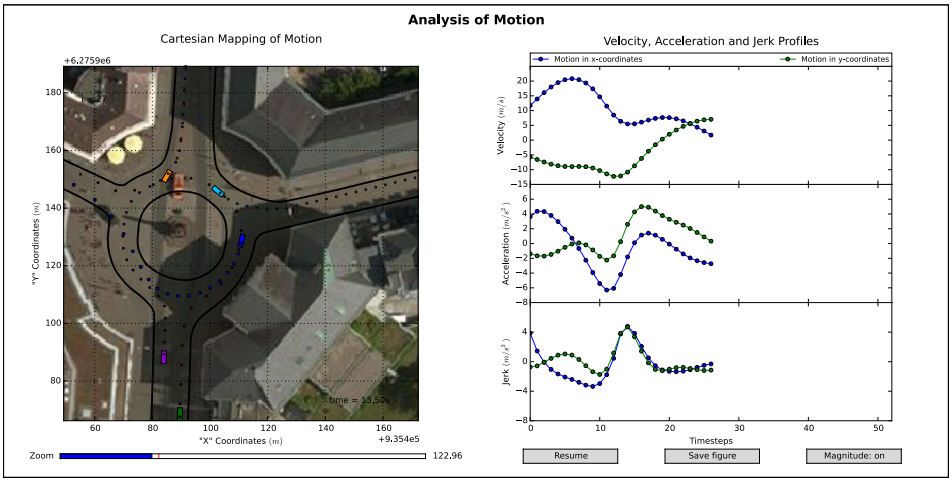
An another combination is to accelerate slightly more and to enter the intersection also before the purple vehicle. This case is illustrated on Figure 7.4c. The acceleration and jerk values while cornering have increased significantly. It should also be noticed that vehicle is not driving on the centerline any more. In order to reduce the acceleration and jerk terms, the planner has exploited the width of the lane while cornering.



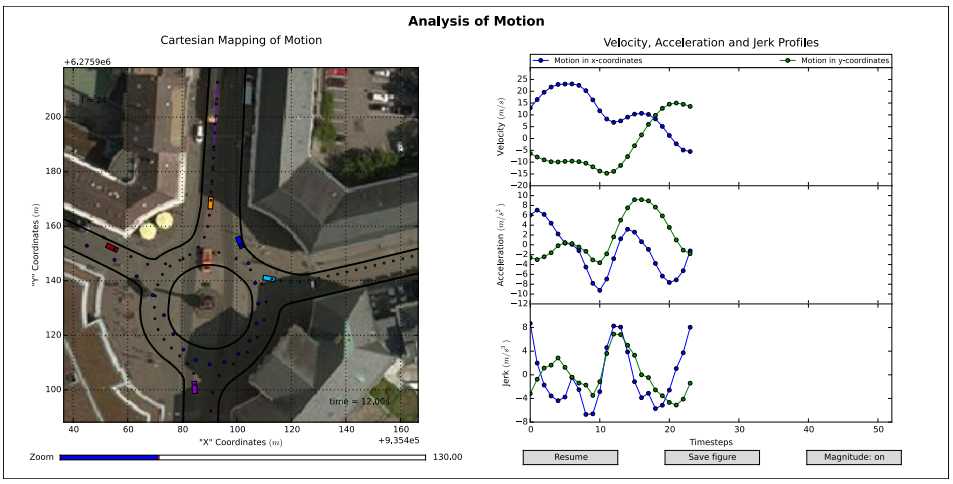
(a) Ideal combination for approaching a roundabout (combination: '10000'). Figure taken at $t = 15.5s$.



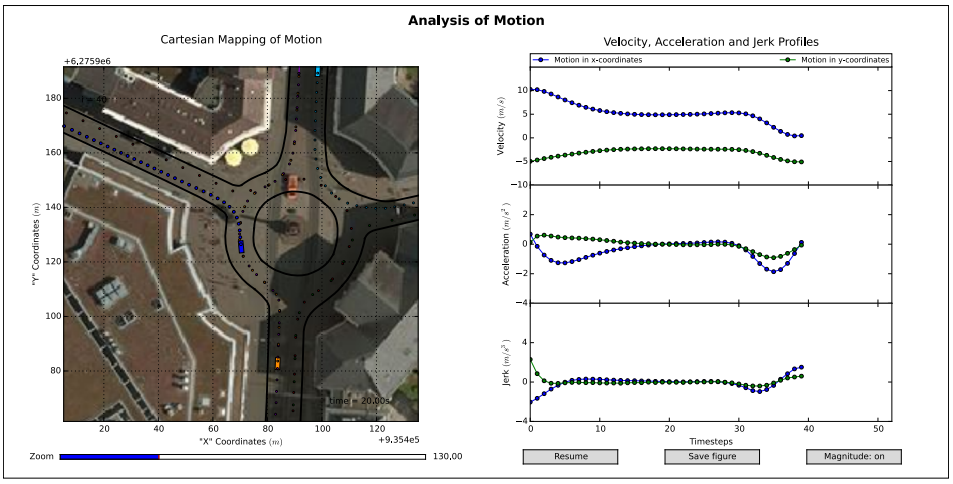
(b) The alternative combination of driving before green vehicle (combination: '10100'). Figure taken at $t = 13.5s$.



(c) The alternative combination of driving before green, and purple vehicle (combination: '10101'). Figure taken at $t = 13.5s$.



(d) The alternative combination of driving before green, purple, and cyan vehicle (combination: '10111'). Figure taken at $t = 12.0s$.



(e) The alternative combination of yielding to all of the other vehicles (combination: 'oooo'). Figure taken at $t = 20.0s$.

Figure 7.4: Distinct combinations for the roundabout at Rondellplatz, Karlsruhe, DE.

A further combination is to accelerate even more and also to lead to the cyan vehicle. This combination is given in Figure 7.4d. The acceleration and jerk values in this case jump to very high values, which might be infeasible due to the physical limits of the vehicle. Another issue to be remarked is the loss of smoothness of the jerk profile. This indicates that the optimization algorithm has not completely optimized the trajectory, *i.e.* further iterations could deliver better results. The selection of relative function tolerance as the optimization termination criteria is the reason behind why such a case is observed. The increase in cost resulted in the denominator given in Equation 3.6b to increase, making the change at each iteration become smaller. This is also the reason behind the jerkiness of the first combination at t-junction example, whose path-time

diagram (PT) was presented in Figure 7.2b.

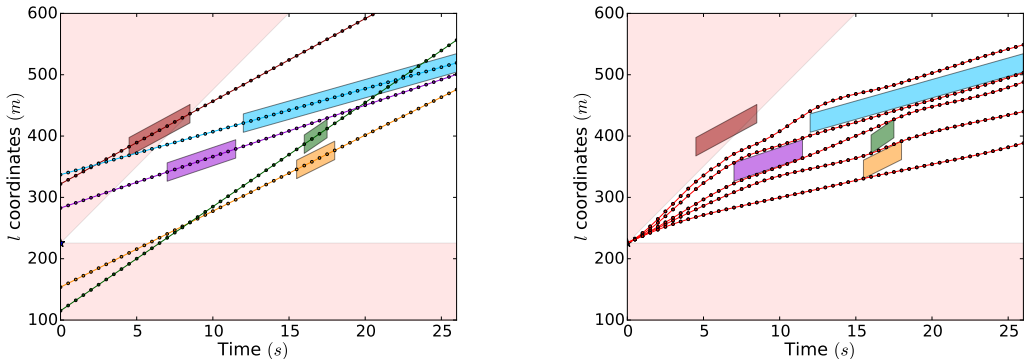
A further combination for the roundabout scenario is to yield all of the vehicles present. The corresponding motion is illustrated in Figure 7.4e. It is evident that the vehicle enters the roundabout towards the end of the horizon. The cornering terms are also very small, the acceleration, for example, does not exceed $2m/s^2$. Quite surprisingly, this combination returns the minimum cost among the others. There are two reasons behind why this is the most favorable one. First of all, as indicated, the cornering values are very small when compared with the other combinations. Apparently, the weights of the cost function have promoted a very comfortable, and also a sluggish ride. Therefore, this combination has resulted in being the most favorable one.

The reason mentioned is obvious. If an agile motion profile is desired, increasing w_{velocity} will cause another combination, for example the one given in Figure 7.4a, to become the most favorable. However, there is a further reason, and this can under some circumstances be a primary matter of concern. In order to illustrate it, a scenario that is based on the same example can be considered. This time, deviating from its original version, an agile motion is desired and the weight factors are appropriately arranged. Additionally, the planning horizon has been reduced to about two third of the previously illustrated value. In such a case, the combination of yielding to orange vehicle will result in a motion profile similar to the previously presented combination of yielding to all vehicles (combination: 'ooooo'). However, as the two third of the previous horizon is selected, the vehicle will not enter the roundabout at all. Hence, the combination for that instant will be free from any penalties resulting from cornering, which eventually causes its cost to be lower than the other combinations. Although an aggressive motion was desired, the sluggish combination has become again the most favorable one again. If w_{velocity} is selected high enough, then that combination will not become favorable any more, but this may result in far too agile motion profiles.

The considered example reveals the complexity while tuning weight parameters. For the stated problem an another remedy can be proposed. It is clear that the trajectory nodes corresponding the beginning of the horizon are more *reliable*. A change in the expected road geometry will cause the position of the trajectory nodes at the beginning of the horizon to change less than the nodes at the end. The reason behind is the accumulation of the change. Hence, it might be beneficial to multiply the nodes with additional weights based on their reliability. Weighting based on Gauss distribution might be a suitable choice. This would also be beneficial for modeling the sensor uncertainty, as the motion of objects at the limits of sensor range is likely to be more unreliable than the objects that are in the vicinity.

Motion profiles given in Figure 7.4c and especially the one on Figure 7.4d have acceleration values which probably exceed the physical limits of many vehicles. During the simulation, the parameters were not constrained by an acceleration constraint. As mentioned in Section 4.1.2, restricting magnitude of acceleration does not accurately represent the physical limits of motion. Therefore, whether the illustrated motion profiles are infeasible or not is ambiguous. Furthermore, an initialization done on a visibility

graph disregards the limits in acceleration. So, even if the internal constraints would be activated, as the initial guess does not guarantee holding the acceleration constraint, the algorithm may not converge to a local minimum. Hence, in the presented figures, high acceleration values were only penalized by an increasing cost term. Although this may appear as an impractical situation, for a reasonable balance between ride comfort and agility, such combinations will return high cost values, eventually making them to become unfavorable.



(a) Motion of other vehicles on PT diagram. **(b)** Alternative combinations on PT diagram.

Figure 7.5: Spatiotemporal analysis of the roundabout considered in Figure 7.4.

During initialization, the feasibility of a combination was determined by two constraints, the velocity constraint and the feasible interval constraint. That is the reason behind why the combination of entering the intersection before the dark red vehicle was not checked. It simply could not satisfy the velocity constraint. This can clearly be seen on a path-time diagram. The path-time diagram of the vehicles present in the environment are given in Figure 7.5a, and the optimized combinations are given on Figure 7.5b. The Initial guess of all of the feasible combinations is also given in Appendix A.3.

Combination	'00000'	'10000'	'10100'	'10101'	'10111'
Cost of initial guess	517.13	339.78	1151.05	1879.57	5466.75
Cost of optimized motion	23.63	40.29	85.56	111.77	294.64

Table 7.2: Costs of alternative combinations for the roundabout scenario in Figure 7.4.

7.2. Comparison of Path-Velocity Decomposition and Complete Planning

In Section 5.3 the path-velocity decomposition was presented as a way for reducing the computational burden of trajectory optimization in kinodynamic environments. It was

mentioned that once the path on which the vehicle would drive is found, by optimizing l-coordinates online, collision free trajectories can be obtained. Now, a comparison on the trajectory qualities between complete planning (CP) and path velocity decomposed (PVD) planning for the narrow pass scenario at the KIT campus will be given.

The scenario illustrated in Figure 7.1c is recomputed via path-velocity decomposition. The resulting trajectory for the driving before cyan and after orange vehicle (combination: '10') is presented in Figure 7.6a. The trajectory obtained required about the same speed with which the path was generated. As a result, the path has served as perfect reference and a very smooth motion profile is obtained. This is also supported by the cost value of that combination at $t = 75.5$, which is given in Table 7.3. There is only a slight difference in between the costs, and therefore the trajectories can practically be considered as identical.

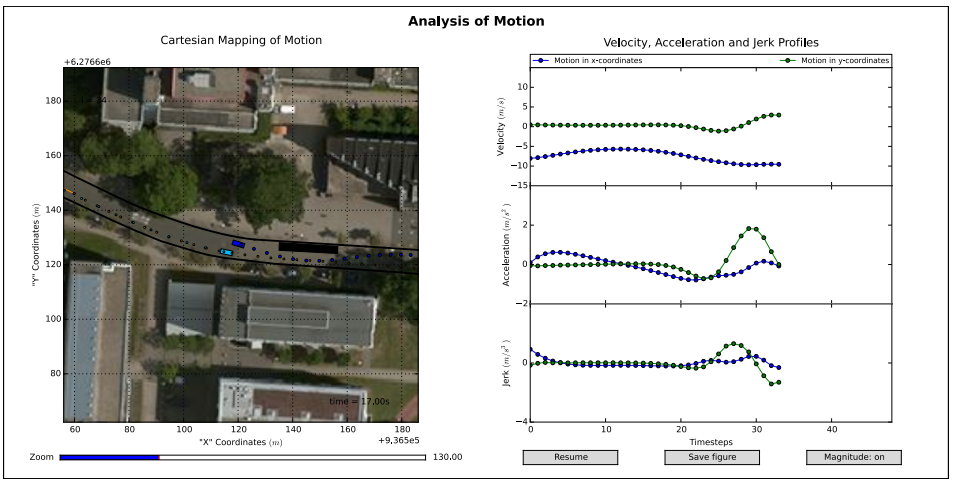
Combination	Timestamp								
	75.5	76.0	76.5	77.0	77.5	78.0	78.5	79.0	79.5
'11'	71.81	89.60	117.97	200.18	–	–	–	–	–
'10'	18.50	18.40	18.19	17.98	17.71	17.37	16.96	16.47	15.90
'00'	42.36	48.98	49.83	50.26	50.57	51.16	51.98	51.92	52.36

Table 7.3: Cost-evolution of feasible combinations throughout the time when computed with PVD. The situation considered is the same as the one in Figure 7.1c.

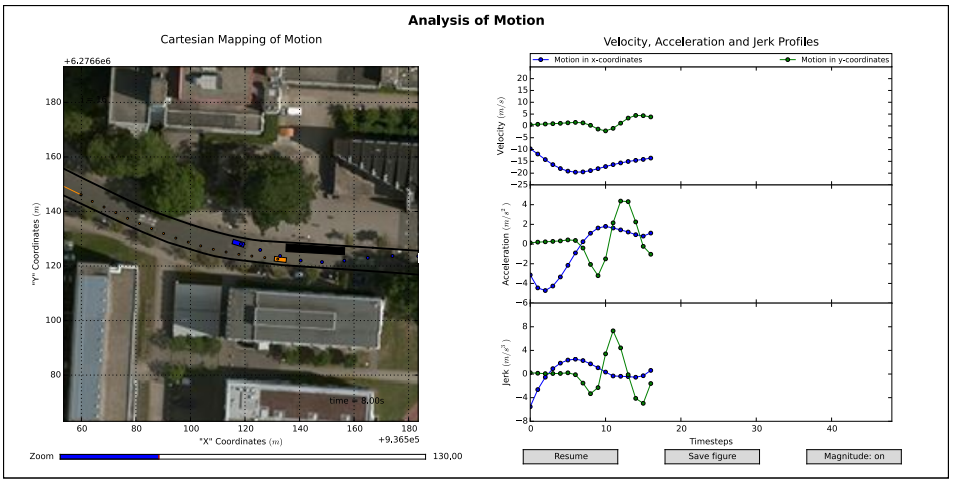
For the same timestamp, the trajectory of the combination driving before both of them (combination: '11') is given in Figure 7.6b. It is evident from the acceleration and jerk diagrams that the trajectory is not smooth any more. The reason behind the jerkiness is the speed of vehicle, which considerably exceeded the level with which the path was generated. This combination returned a cost value of 71.81, whereas the complete planning had returned only 46.46.

In order to reveal the factor that lead to a reduction in costs, the same combination, but this time computed via complete planning, is given in Figure 7.6c. If the Cartesian mapping of motion is inspected, it will be realized that the vehicle has driven closer to the black obstacle, and thereby has sacrificed from following the centerline, but in contrast, has gained a lot in terms of ride comfort.

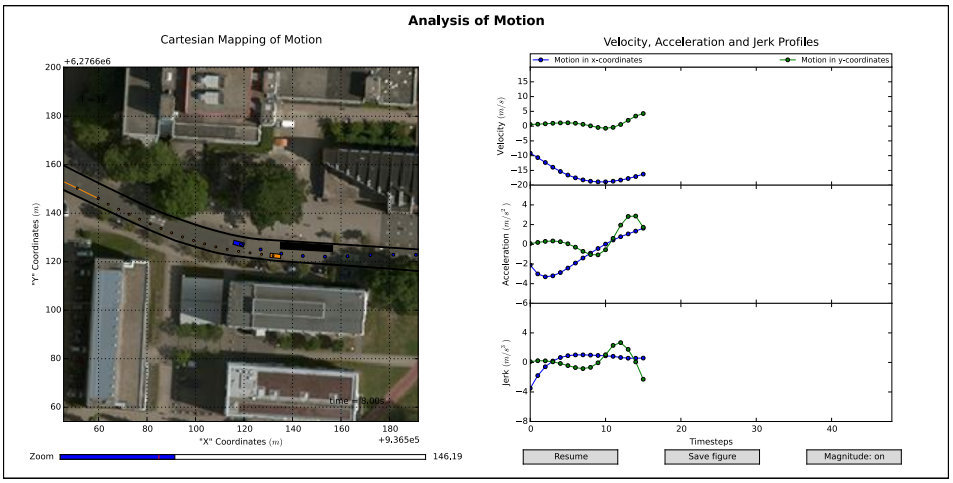
The cost values from timestamp 75.5 on show the same trend. The difference in costs becomes more prominent when the vehicle drives at higher speeds than with which the path was planned. On the other hand, if the speed is lower than the level of path planning, the difference between complete planning and velocity decomposed planning is negligible. In terms of computational cost, the number of required function evaluations is in general about one third to one fourth of in complete planning. As a final remark,



(a) PVD of a combination that requires speeds at about the level with which the path was generated. The smooth motion profile is remarkable.



(b) PVD of a combination that requires higher speeds than with which the path was generated. The motion diagram is not smooth any more.



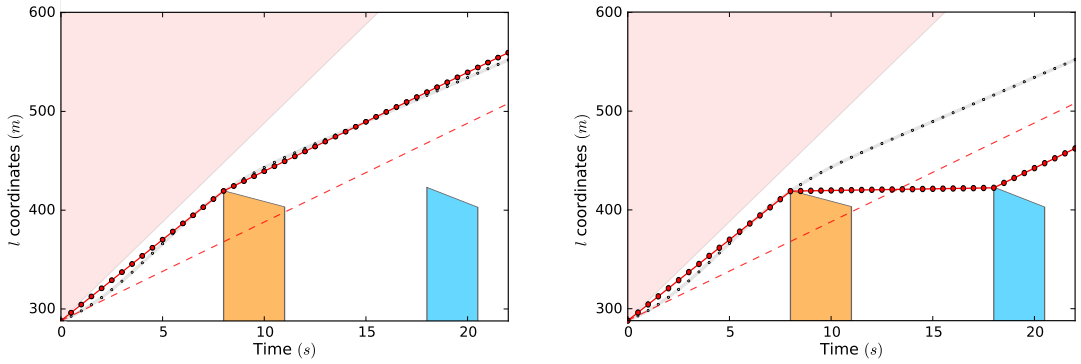
(c) The same combination as illustrated in (b) however, this time planned with CP. It should be remarked that the vehicle gets closer to the black obstacle while crossing the passage.

Figure 7.6: Comparison of complete and decoupled-planning for a narrow pass.

as discussed on Page 69, because the *d*-coordinates cannot be utilized, escape maneuvers from obstacles, and also overtaking if it is founded that, cannot be performed via path-velocity decomposition.

7.3. Influence of Initialization on Optimization Process

The significance of initialization on the convergence was indicated many times throughout the thesis. The solver used, namely the SQP method, is a Newton-type solver and the foun minimum and convergence of these methods depend highly on the initialization done. If the initialization of a Newton method is 'close enough' to the solution, then quadratic convergence can be achieved. Relying on this information, it may be claimed that, the initialization process introduced in Section 5.2.3 is unnecessary, as the initialization obtained through the use of basic visibility graph is already imitating the solution good enough. Subsequently, due to the resulting quadratic convergence, the benefits delivered by the improved visibility graph approach is negligible. The deficiency of path-time diagram at representing road geometry may be addressed as a contributing factor to the claim.



(a) *Good* initialization.

(b) *Bad* initialization.

Figure 7.7: A comparison of *good* and *bad* initialization. The red dotted line indicates the initialization and the grey line behind represents the optimal solution.

In order to inspect the influence of initialization, a problem is once initialized with the improved visibility graph and then with the basic visibility graph approach. In Figure 7.7, initializations performed for the combination '11' of the narrow pass scenario at the KIT is given as an example. The red dotted line on the figures indicates the initial guess and the grey line at the background represents the optimal solution. The initialization obtained from the improved visibility graph is given on the left side, and the one obtained from the basic visibility graph is given on the right side. According to the path-time diagram, the improved visibility graph mimic the solution very good,

whereas the basic visibility can only poorly imitate the solution. A comparison is performed both for complete planning and velocity decomposed planning. The results are presented on Table 7.4.

	CP		PVD	
	Good	Bad	Good	Bad
Number of function evaluations	390	421	255	209
Optimized value	40.95	40.93	60.18	60.18

Table 7.4: Comparison of *good* and *bad* initializations.

Values in the table reflect that there is not any substantial difference between the initializations. They converged to the same minimum after roughly equal number of function evaluations. In the case of decomposed planning, the 'bad' initialization has even returned the solution faster. Although this example implies that the initial guess obtained from the basic visibility graph was already close enough to the solution, this single examples is not adequate for a generalization.

Initialization does not only have an influence on the convergence, but also on where the solution will converge to. A local optimization algorithm has the potential to get stuck to a stationary point, and to converge to an undesirable minimum. This danger can be eliminated either by constraints, or by a close to minimum initialization. Considering the previous example, if the optimization problem is badly initialized and if backwards driving is not prohibited by a constraint, a motion profile given in Figure 7.8 can be obtained. This motion profile in the figure evidently corresponds to a profile other than the desired minimum.

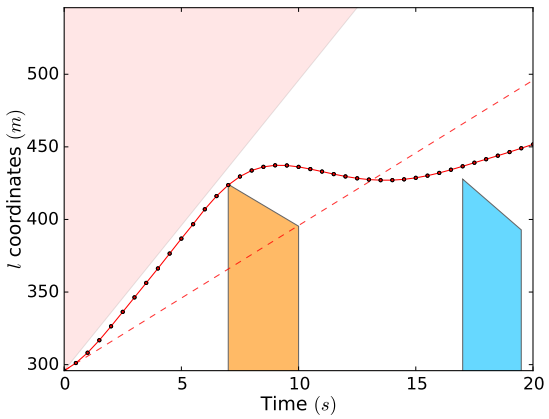


Figure 7.8: PT diagram of a badly initialized problem. Due to the absence of necessary constraints, the initialization has led the algorithm to converge a point any other than the desired minimum.

The endeavor for obtaining a close to minimum initialization brings about a further advantage. A variety of distinct analysis results have proven that, the combination whose initialization returns the smallest cost generally remains the most favorable one at the end of the optimization. Capitalizing on that, a perfect heuristic for determining which of the combinations does probably yield the most favorable solution can be established. Whenever the number of feasible alternative combinations exceed the limits of computational power, this heuristic can be utilized and then, only the combinations that probably would yield the minimum will be optimized.

Finally, as the visibility graph ignores the inertia of the vehicle and allows instantaneous changes in speed, it will yield poor initial guesses if the weight factors are adjusted for a comfortable ride. A remedy in this case could be to use a *cascade optimization* structure. First, based on the initialization obtained from the visibility graph, path-velocity decomposed trajectory planning along the road centerline can be performed. Once a result from the one dimensional optimization obtained, it can be served as the initial guess to the complete planner. This will yield an efficient and close to minimum initialization. The initialization done in this way will also include the geometry of the road, making the approach remarkably suitable for windy roads.

7.4. Convergence Process of the Optimization Algorithm

The convergence process founds the core of trajectory optimization. Monitoring the progress of the solver algorithm plays an essential role on understanding the shape of the cost function and the behavior of line search algorithm. In this respect, the progress of many scenarios has been inspected. Some of the generalizations inferred are presented next.

Section 6.4 indicated that the simulations are performed by selecting the relative function tolerance as the termination criteria. In this terms, obtaining smooth jerk profiles for a planning horizon of 15-24 seconds at mostly straight roads required tolerances reaching $1e^{-13}$. This typically corresponds to node computation precision higher than 10cm, and function evaluations in between 1000-1500. Certainly, the number of evaluations depend on how strict the problem is constrained. However, for trivial problems, such as the ones illustrated so far, after $500-600$ evaluations smooth velocity and acceleration profiles are generally obtained. The rest of the evaluations provide smoothness of the jerk profiles. In terms of relative function tolerance, this typically corresponds to values ranging from $1e^{-5}$ to $1e^{-8}$.

In fact, the convergence rate decreases significantly after a tolerance value of $1e^{-10}$, or expressed in terms of function evaluation, after 800 evaluations. This issue is illustrated in Figure 7.9. In the first figure, the optimized motion profile and the cost for the 1000^{th} evaluation is given. As printed in the GUI, the reached tolerance is $1e^{-12}$, the optimum

cost value is 31.23878834, and the cost of the tested value at that instant is 31.23883449. In the second figure the situation after 1500^{th} evaluation is given. The reached tolerance and the minimum cost are still the same. At that instant, the cost of the tested profile is 31.23883449. This situation implies that the algorithm jumps over the minimum, from one side of the valley to the other side of it. The fundamental reason why this occurs is the shape of the problem. The cost values require a tolerance value at about the level of numerical precision. Due to the approximated Hessian and the numerical gradient calculation, numerical errors become dominant and plague the computation.

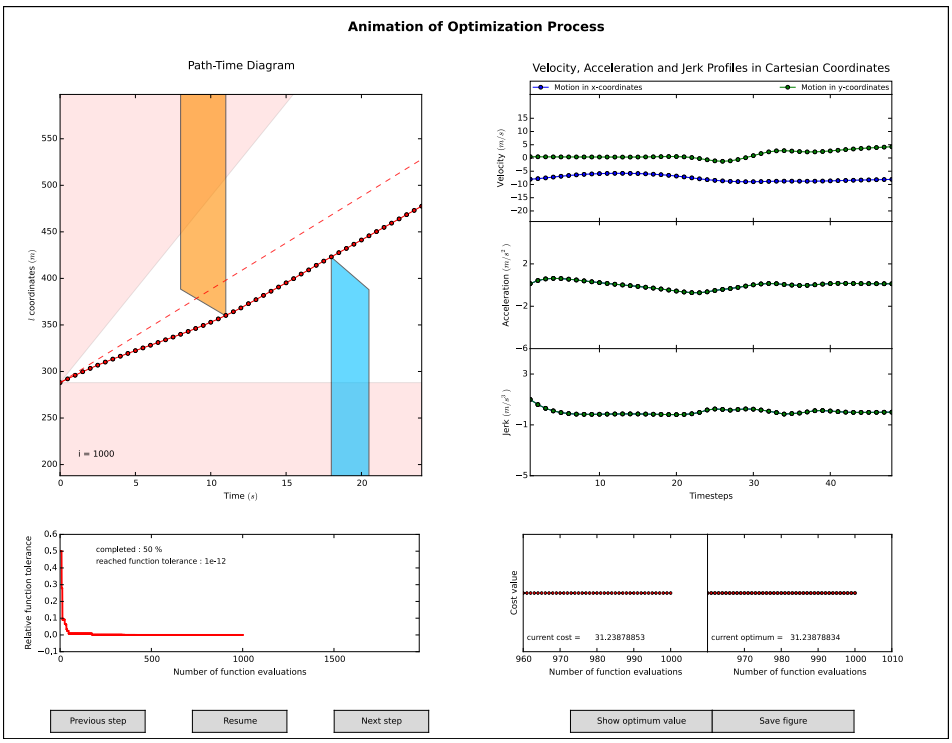
A further issue to be inspected is the effect of scaling. Different locations around the world have from each other significantly varying x and y-coordinate values. This evokes the analysis of the optimization algorithms vulnerability to prescaling. In this respect, a naive test was done. The initial coordinates of the ego vehicle is subtracted from the coordinates of all of the objects, including the roads defined in the environment. The solution process is then compared with the conventional approach. Although this does not resemble a correct analysis of scaling, it gives a rough estimate of its influence on the optimization problem. The results obtained did not exhibit a remarkable difference in the number of function evaluations between the both cases. It can be concluded that, the effect of scaling on the optimization process is not prominent.

7.5. Further Remarks on Results

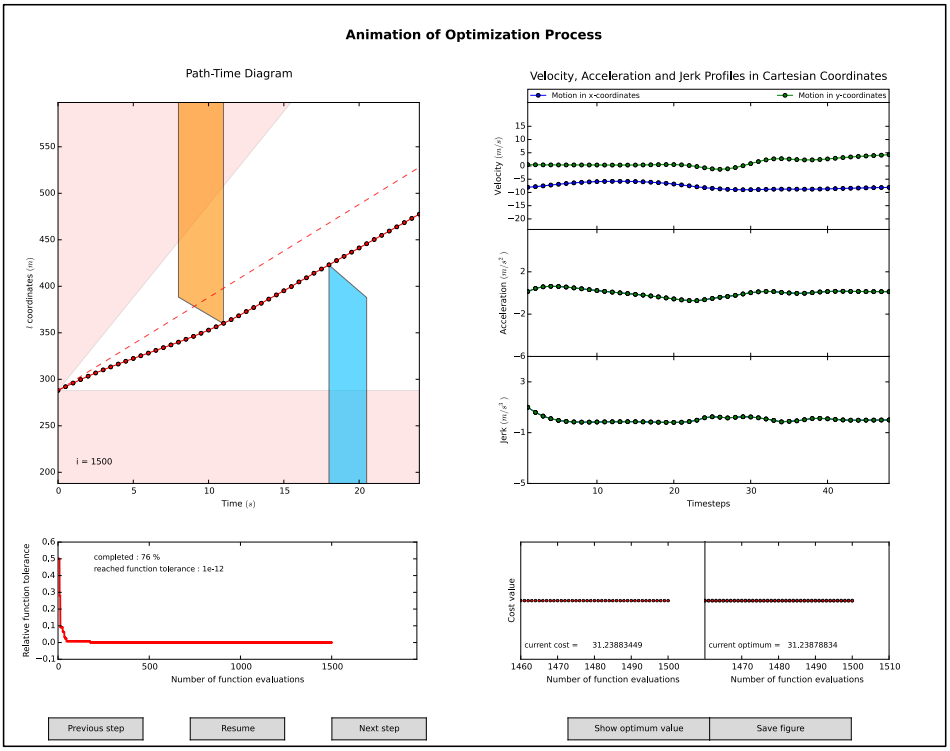
There are several further remarks that do not fit into the previous sections, but are advantageous to convey. These will be presented here.

The analysis has revealed that, regardless of the motion of other obstacles, the number of alternative combinations depend on the sensor range of the ego vehicle and the length of the planning horizon. The reliably delivered sensor range is a vital factor on the trajectory prediction of other vehicles. If it is plagued by remarkable errors, fallacious reference trajectories can be generated. Besides the sensor range, also the planning horizon is a decisive factor on the evaluation of alternative combinations. Any combinatorial option must be covered by the range of planning horizon. If the planning horizon is not long enough, the motion profile required to maintain a combination will exceed the internal limits of the vehicle. This will eventually make many combinations to become infeasible. A combined planning length and sensor range of at least 8-10 seconds will allow the evaluation of several alternatives. At low speeds, for values in the range of 4-5 seconds only a few options will be feasible.

Speed, acceleration etc. values were calculated for motion in Cartesian coordinates. Underlying reason behind it was the fact that the Frenet coordinates do not discriminate between curved and straight roads. The analysis results have shown that, the values obtained from Frenet and Cartesian systems can deviate about 20%.



(a) Optimization process at 1000th evaluation.



(b) Optimization process at 1500th evaluation.

Figure 7.9: Optimization process presented after two different evaluation numbers.

The optimization library NLopt offers two local, derivative based solvers for inequality constrained problems. The SLSQP and the 'Method of Moving Asymptodes' (MMA) [64]. Both of them are Newton-type solvers and approximate the Hessian. Because the SQP algorithms are renowned for their success at handling nonlinear problems, the SLSQP was utilized. However, several scenarios were also solved with the MMA. In general, the MMA performed 4-5 times slower than the SLSQP. The SLSQP exhibits very *aggressive* step size selection when compared with other algorithms. It easily jumps to the other edge of the cost-contour valley. This may be the underlying reason of its speed of convergence. However, such long and aggressive step lengths also increases the danger of getting stuck to a stationary point around the minimum, if such points are present.

The optimized results for the indicated tolerance values are obtained roughly after 700-1000 function evaluations. When the simulation program was run with a local, derivative-free optimization algorithm, such as with COBYLA (Constrained Optimization by Linear Approximations) [65], 1000 evaluations were reached after one minute. Due to the numerical derivative approximation 1000 evaluations took considerably longer than that. However, if the gradient and the Hessian is provided analytically, the SQP converges much more faster and rarely needs to exceed two digit number of evaluations. In such a case, the probability of getting trapped into a stationary point is also less than the case in approximated Hessian.

8. Conclusion and Future Work

This work has demonstrated the integrability of combinatorial reasoning into continuous methods for optimal motion planning. For obstacle free environments, it is shown that globally optimal trajectories can be generated by accurately initializing an appropriately established optimization problem. In structured kinodynamic environments, whenever the ego vehicle has to merge into traffic flow at a non-signalized intersection, or pass with the oncoming traffic, various combinations on sequencing occur. When reasoning among the set of alternative combinations is modeled as an optimization problem, each of the feasible combinations correspond to convex regions, having distinct local minima. An option for attaining the global minimum among them would be to use global optimization algorithms. However, these methods lack real time implementability. As a remedy, the problem is subdivided in accordance with the combinatorial options by exploiting inequality constraints. Afterwards, feasible trajectories for the individual combinations are computed by a Newton-type local optimization routine. Once the costs of alternative combinations are achieved, the one, returning the smallest costs is selected. Thereby, the globally optimal trajectory is procured.

As an indispensable requirement of utilizing a local optimization algorithm, the solver is fed with an initial guess, that meets all of the constraints set on motion. Yielding an initialization that is inside the solution set and resembles a good estimate of the solution has been a primary concern throughout the thesis. In order to reduce the load of the computation, the infeasible combinations are eliminated at the level of initialization and are not sent to the solver routine. Therefore, besides the aforementioned requirements, the detection of feasible combinations has also emerged as an additional task onto the initialization. Considering these aspects, the basic visibility graph approach has been improved and subsequently utilized on path-time space. In general, the initial estimation done in this way imitated the solution good enough. However, two significant drawbacks of the method has appeared. First, as a result of being based on visibility graph approach, the initialization allows instantaneous changes of speed values. Second, as the initialization is performed on path-time space, the geometry of the road is not considered.

The results have proven that the presented approach of dividing the combinatorial problem into local optimization problems is applicable to any type of scenarios. Truly optimal trajectories for various scenarios from all around the world have been successfully generated. It is further demonstrated that the presented approach serves a basis for the study on overtaking scenarios. Comparisons made between complete planning

and path-velocity decomposed planning have revealed that, when driven at about the speeds with which the path was generated, the costs of path velocity decomposed trajectories do not remarkably deviate from costs of the trajectories created with complete planning. However, as the path-velocity decomposition plans a motion along a given path, and does not utilize lateral coordinates, it is ineligible for realizing overtaking and escaping from obstacles.

The initializations obtained from the improved visibility graph have proven to be close enough to the optimal trajectories. Moreover, among combinatorial options, the one whose initialization delivers the least cost has remained the most favorable one after the optimization. This suggests that, if not a very comfortable ride is sought for, the improved visibility graph could be used as a perfect heuristics for determining which combinations are likely to return the best combination. Exploiting from such an heuristic would be very beneficial whenever the computational power is scarce and a number of feasible alternative combinations are available. As the visibility graph approach is incapable of inspecting feasibility in terms of acceleration, the acceleration constraint was deactivated. Hoever, as a result of exploiting an energy-type cost functional, the combinations requiring high acceleration values return increasing cost values, which eventually makes them unfavorable among others. Considering that the combination of preserving the current sequence will be available at all times, such a penalization based approach also does not pose any safety-critic problems.

During the simulations, the analytical gradient and Hessian was not provided to the solver algorithm. Therefore, the gradient was calculated numerically and the Hessian was approximated. With this setting, smoothness in jerk was maintained after 700-800 function evaluations. If the analytical counterparts would be used, the same results would be reached after 20-30 evaluations. Certainly, when more evaluations are required, the danger of encountering runtime errors and reaching round-off limits typically increases. Motion planning based on local continuous methods require a strong intuition and vast experience about the shape of the contour lines and the structure of the optimization problem. Boyd and Vandenberghe (2009) describe this phenomenon on local optimization with the following: "*Roughly speaking, local optimization methods are more art than technology. Local optimization is a well developed art, and often very effective, but it is nevertheless an art*" [40, p. 9].

It has further been revealed that, two interconnected factors play a major role on the detection of alternative feasible combinations. The first one is the sensor range with high reliability. The other factor is the planning horizon of the ego vehicle. Both of them determine feasible region of the obstacle-relative motion, and depending on it, the number of alternative combinations. Currently, the values supplied by state-of-art autonomous vehicles are only likely to perform combinatorial reasoning at low speeds and find a very limited amount of feasible alternatives.

Future work could include deriving analytical gradients and Hessians and supplying them to the optimization problem. The deficiency of the initialization can be resolved by utilizing either sampling based methods, or the cascade optimization proposed on

Page 79. As introduced in Chapter 2, a variety of constraints can be employed on sampling based methods. Capitalizing on that, an initial guess that is inside the solution set can be obtained. If these methods would be utilized, the obstacles on path-time diagram will not be necessarily polygonal obstacles any more. This will facilitate the adjustment of intervehicular safety distance at varying relative speeds, allowing the trajectory prediction of other vehicles to involve their instantaneous acceleration. While considering the problem the intersections were assumed to be non-signalized. However, the traffic lights for example can also be treated as a source of combinatorial options and be included inside the analysis. The traffic signs on the other hand can also be modeled as a constraint and imposed into the optimization problem. If they are implemented, more realistic scenarios would be obtained. As a further simplifying assumption, the obstacle behavior was accepted to be perfectly known and the sensor uncertainty was neglected. Studies to abandon this assumption can be done. For example, the method presented on Page 73 can be utilized for dealing with uncertainties. Finally, the altitude information can be included into the geometrical mapping framework, and by involving it into the optimization process, energy optimal trajectories can be obtained.

A. Appendix

A.1. Settings File

```
settings = {
    "Main" : {
      "horizon"                    : 12,     # s
      "dt"                         : 0.5,    # s (step-width)
      "speed_limit"                : 20,     # m/s
      "acceleration_limit"         : 2.5,    # m/s2
      "desired_travel_spd"         : 10,     # m/s
      "distance2static_obs"        : 1.0,    # m
      "v2v_safety_dist"            : 15.0,   # m
      "Path-Velocity_decompostion" : False,
      "travel_length"              : 0.5     # s
    },
    "Map" : {
      "road_data"                          : 'KIT',
      "centerline_computation_step-width"  : 5,
      "centerline_computation_tolerace"    : 1e-12,
      "global_coordinates"                 : True,
      "verbose"                            : True
    },
    "Opt" : {
      "weight_factors"   : {
                           "velocity"     : 10,
                           "acceleration" : 25,
                           "jerk"         : 100,
                           "track"        : 500
                          },
      "constraint_tol"   : 1e-12,
      "relative_tol"     : 1e-12,
      "method"           : 40,
      "constraints"      : ["roadsides_corners",
                           "speed_limit",
                           "collision_avoid",
                           "acceleration_limit"],
      "verbose"          : True,
      "trace"            : True
    },
    "Opt_1D" : {
      "weight_factors"   : {
                           "velocity"     : 10,
                           "acceleration" : 20,
                           "jerk"         : 400,
                           "track"        : 500
                          },
      "constraint_tol"   : 1e-8,
      "relative_tol"     : 1e-8,
      "method"           : 40,
      "verbose"          : True,
      "trace"            : True
    }
}
```

A.2. Pseudocode

Algorithm 1 Curvature Adaptive Centerline Computation

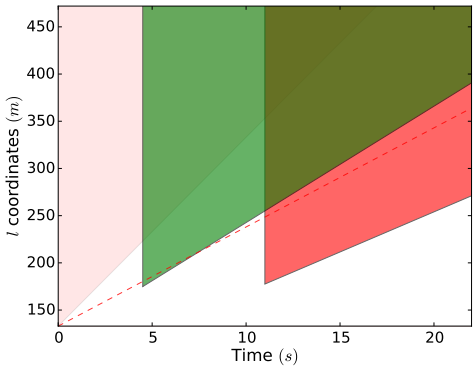
Input: reference curve, d-value, array of equidistant l-coordinates
Output: curvature adaptive l-coordinates

1: find Cartesian coordinates for each of the l_i (using predefined d-value and pseudodistance object of the reference curve).
2: create an array whose first column stores the l-coordinates l_i and in its second column the angle-difference ϕ_i found in Cartesian coordinates

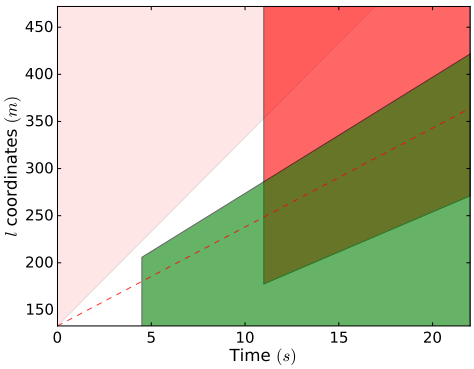
3: $i = 1$
4: **while** $i < \text{len}\,(\text{array}) - 1$ **do**
5: **if** ($|\phi_{i+1} - \phi_i| > \text{threshold}_\phi$) $\wedge$ ($|l_{i+1} - l_i| > \text{threshold}_l$) **then**
6: Add a new point between i and $i + 1$
7: recalculate the array
8: **else**
9: $i = i + 1$
10: **end if**
11: **end while**

12: **return** the first column of the array

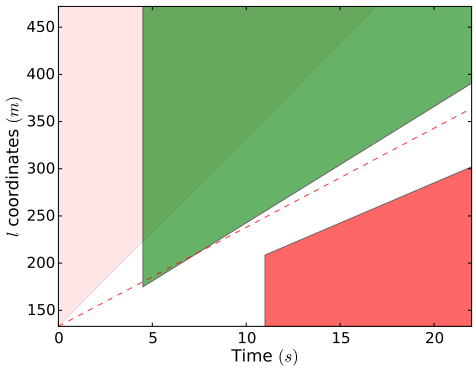
A.3. Supplementary Figures



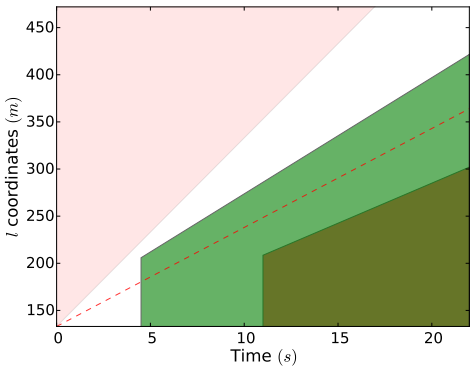
(a) Combination ‘00’.



(b) Combination ‘01’.

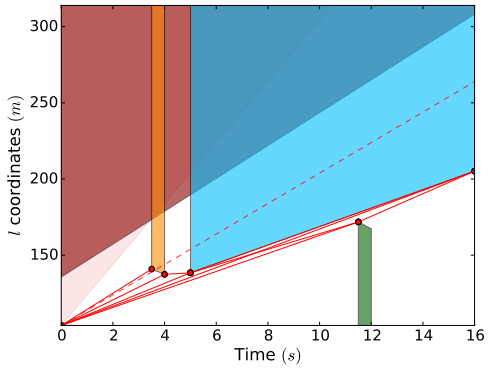


(c) Combination ‘10’.

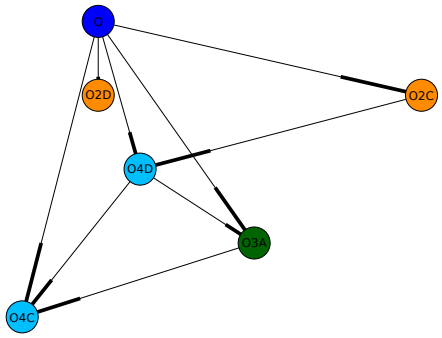


(d) Combination ‘11’.

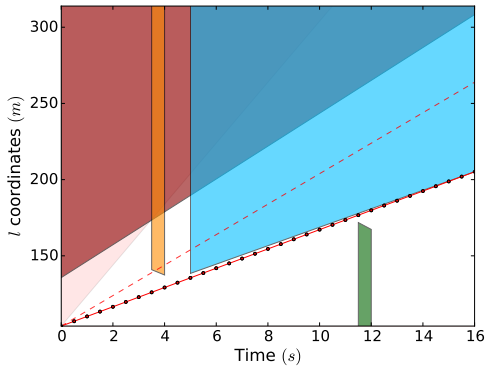
Figure A.1: All of the possible combinations on PT graph. Infeasibility of the combination ‘01’ is apparent.



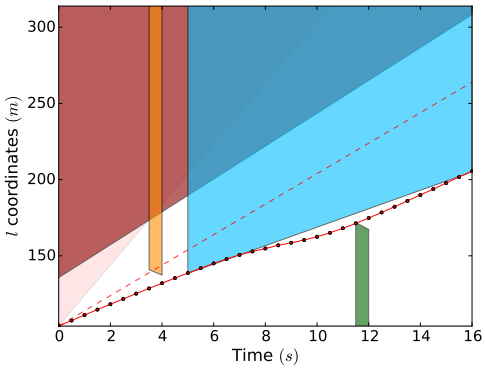
(a) Visibility diagram.



(b) Corresponding search tree.

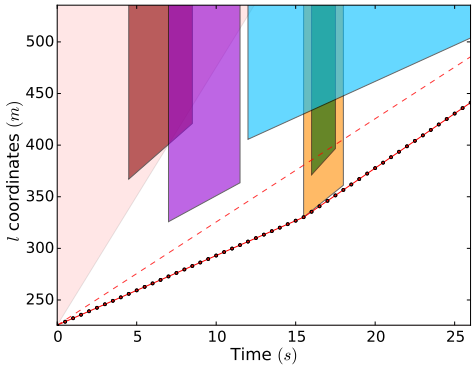


(c) Performed initial guess.

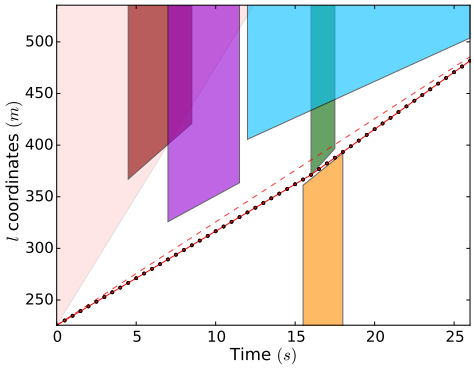


(d) Optimized trajectory.

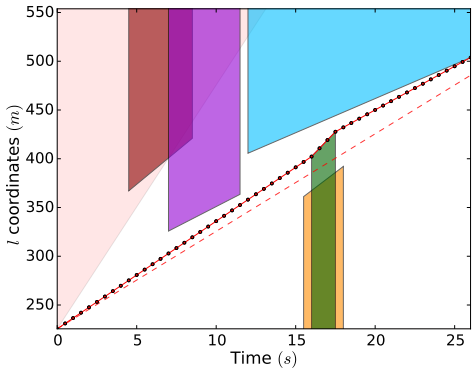
Figure A.2: Steps of initialization for the combination '1000' in Figure 7.1b.



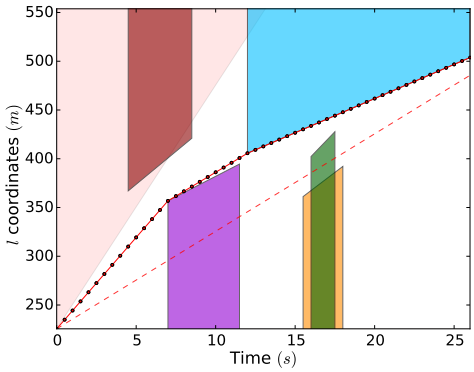
(a) Combination ‘00000’.



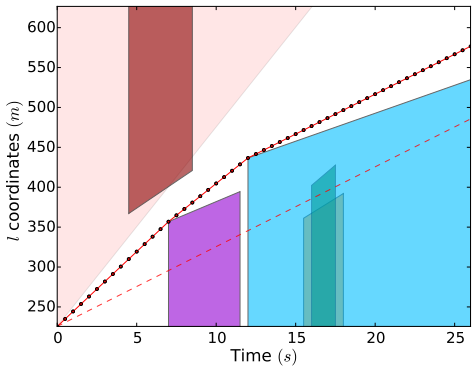
(b) Combination ‘10000’.



(c) Combination ‘10100’.

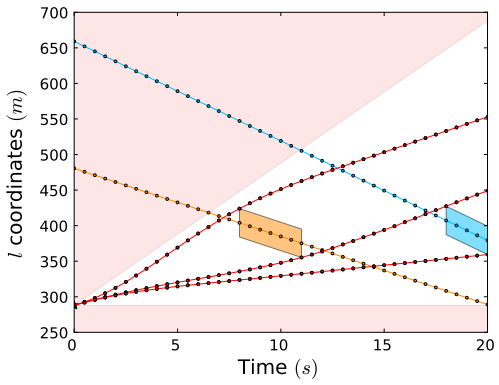


(d) Combination ‘10101’.

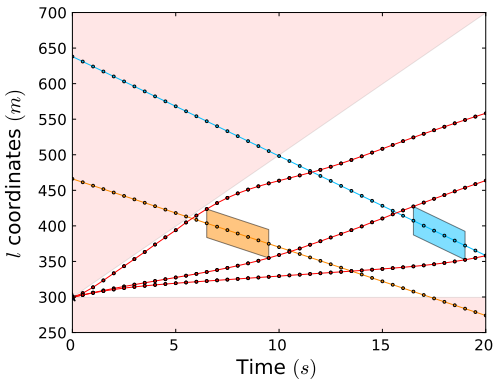


(e) Combination ‘10111’.

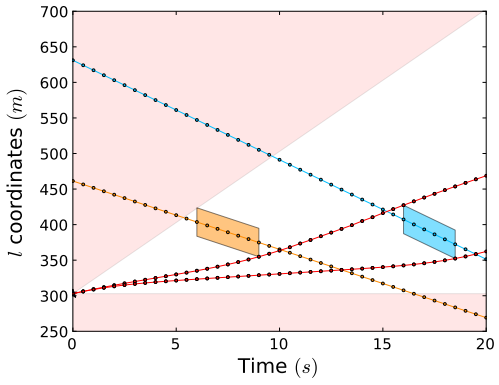
Figure A.3: Initialization of feasible alternative combinations for the roundabout scenario considered in Figure 7.4.



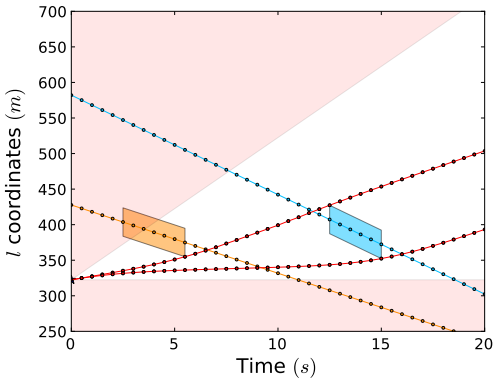
(a) PT diagram at $t = 75.5s$.



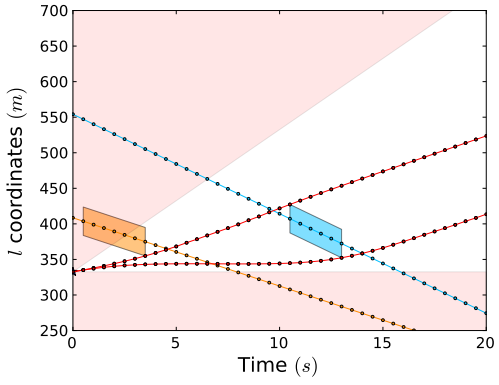
(b) PT diagram at $t = 77.0s$.



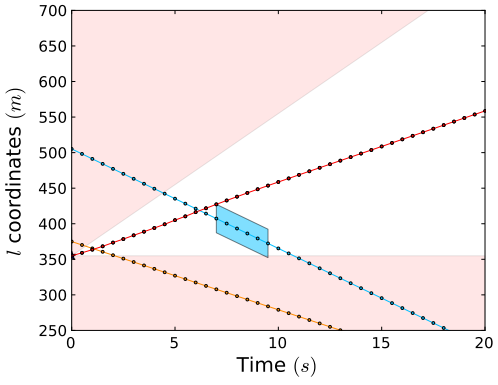
(c) PT diagram at $t = 77.5s$.



(d) PT diagram at $t = 81.0s$.



(e) PT diagram at $t = 83.0s$.



(f) PT diagram at $t = 86.5s$.

Figure A.4: Alternative combinations in Figure 7.1c along time on path-time diagram. In Figure (a), the combination at center is the most favorable one. Therefore, it is selected as the reference trajectory. The first combination, which corresponds to being the lead vehicle, becomes infeasible at $t = 75.5s$. So does also the combination of 'yielding to all others'. At time $t = 86.5s$ that combination is not valid anymore.

Bibliography

[1] Lino Guzzella. "Automobiles of the future and the role of automatic control in those systems." In: *Annual Reviews in Control* 33.1 (2009), pp. 1–10.

[2] Bernhard Ebel and Markus B Hofer. *Automotive Management - Strategie und Marketing in der Automobilwirtschaft*. Springer, 2014. ISBN: 978-3-642-34067-3.

[3] Rajesh Rajamani. *Vehicle dynamics and control*. Springer, 2011.

[4] T. Toroyan. *Global status report on road safety 2013: supporting a decade of action*. World Health Organization. 2013. URL: http://www.who.int/violence_injury_prevention/road_safety_status/2013/en/ (visited on 03/17/2014).

[5] William F Powers and Paul R Nicastri. "Automotive vehicle control challenges in the 21st century." In: *Control engineering practice* 8.6 (2000), pp. 605–618.

[6] Karel A Brookhuis, Dick De Waard, and Wiel H Janssen. "Behavioural impacts of advanced driver assistance systems–an overview." In: *European Journal of Transport and Infrastructure Research* 1.3 (2001), pp. 245–253.

[7] Jennifer. N. Dang. *Updated Estimates of Fatality Reduction by Electronic Stability Control*. Evaluation Note DOT HS 809 790. National Highway Traffic Safety Administration, Sept. 2004. URL: http://www.nhtsa.gov/cars/rules/regrev/evaluate/809790.html (visited on 03/17/2014).

[8] *Regulation 661/2009/EC*. European Parliament and Council of the European Union. Brussels, Belgium, 2009.

[9] *Fatality Analysis Reporting System (FARS) encyclopedia*. National Highway Traffic Safety Administration. 2014. URL: http://www-fars.nhtsa.dot.gov/Main/index.aspx (visited on 03/20/2014).

[10] *Statistisches Jahrbuch 2008 - Deutschland und Internationales*. German Federal Statistical Office / Statistisches Bundesamt, ISBN: 978-3-8246-0822-5. Wiesbaden, Germany, 2008.

[11] Patrick Planing. *Innovation Acceptance - The Case of Advanced Driver Assistant Systems*. Springer, 2014. ISBN: 978-3-658-05004-7.

[12] Julius Ziegler et al. "Making Bertha Drive? An Autonomous Journey on a Historic Route." In: *Intelligent Transportation Systems Magazine, IEEE* 6.2 (2014), pp. 8–20.

[13] Joseph Funke et al. "Up to the limits: Autonomous Audi TTS." In: *Intelligent Vehicles Symposium (IV), 2012 IEEE*. IEEE. 2012, pp. 541–547.

[14] Steven E Shladover et al. "Automated vehicle control developments in the PATH program." In: *Vehicular Technology, IEEE Transactions on* 40.1 (1991), pp. 114–130.

[15] Levent Guvenc et al. "Cooperative adaptive cruise control implementation of team mekar at the grand cooperative driving challenge." In: *Intelligent Transportation Systems, IEEE Transactions on* 13.3 (2012), pp. 1062–1074.

[16] *Statistisches Jahrbuch 2013 - Deutschland und Internationales*. German Federal Statistical Office / Statistisches Bundesamt, ISBN: 978-3-8246-1007-5. Wiesbaden, Germany, 2013.

[17] Bruce Donald et al. "Kinodynamic motion planning." In: *Journal of the ACM (JACM)* 40.5 (1993), pp. 1048–1066.

[18] Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[19] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[20] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. "Real-time motion planning for agile autonomous vehicles." In: *Journal of Guidance, Control, and Dynamics* 25.1 (2002), pp. 116–129.

[21] John Canny. *The complexity of robot motion planning*. MIT press, 1988.

[22] Yoshiaki Kuwata et al. "Motion planning for urban driving using RRT." In: *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE. 2008, pp. 1681–1686.

[23] Kristijan Macek, M Becked, and Roland Siegwart. "Motion planning for car-like vehicles in dynamic urban scenarios." In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE. 2006, pp. 4375–4380.

[24] Romain Pepy, Alain Lambert, and Hugues Mounier. "Path planning using a dynamic vehicle model." In: *Information and Communication Technologies, 2006. ICTTA'06. 2nd*. Vol. 1. IEEE. 2006, pp. 781–786.

[25] Moritz Werling. *Ein neues Konzept für die Trajektoriengenerierung und-stabilisierung in zeitkritischen Verkehrsszenarien*. Vol. 34. KIT Scientific Publishing, 2011.

[26] Christopher R Baker, David I Ferguson, and John M Dolan. "Robust mission execution for autonomous urban driving." In: *Robotics Institute* (2008), p. 178.

[27] Maxim Likhachev and Dave Ferguson. "Planning long dynamically feasible maneuvers for autonomous vehicles." In: *The International Journal of Robotics Research* 28.8 (2009), pp. 933–945.

[28] Martin Rufli and Roland Siegwart. "On the design of deformable input-/state-lattice graphs." In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 3071–3077.

[29] Jean-Paul [Hrsg.] Laumond, ed. *Robot motion planning and control*. Lecture notes in control and information sciences ; 229. London: Springer, 1997. ISBN: 3-540-76219-1.

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

[31] J-P Laumond et al. "A motion planner for nonholonomic mobile robots." In: *Robotics and Automation, IEEE Transactions on* 10.5 (1994), pp. 577–593.

[32] Ryo Takei et al. "A practical path-planning algorithm for a simple car: a Hamilton-Jacobi approach." In: *American Control Conference (ACC), 2010*. IEEE. 2010, pp. 6175–6180.

[33] 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*. IEEE. 2009, pp. 1879–1884.

[34] Julius Ziegler et al. "Trajectory planning for Bertha—A local, continuous method." In: *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*. IEEE. 2014, pp. 450–457.

[35] David G Hull. "Conversion of optimal control problems into parameter optimization problems." In: *Journal of Guidance, Control, and Dynamics* 20.1 (1997), pp. 57–60.

[36] John T Betts. "Survey of numerical methods for trajectory optimization." In: *Journal of guidance, control, and dynamics* 21.2 (1998), pp. 193–207.

[37] Sören Kammel et al. "Team AnnieWAY's autonomous system for the 2007 DARPA Urban Challenge." In: *Journal of Field Robotics* 25.9 (2008), pp. 615–639.

[38] Dawn Tilbury, Richard M Murray, and S Shankar Sastry. "Trajectory generation for the N-trailer problem using Goursat normal form." In: *Automatic Control, IEEE Transactions on* 40.5 (1995), pp. 802–819.

[39] Michael J Van Nieuwstadt and Richard M Murray. "Real time trajectory generation for differentially flat systems." In: (1997).

[40] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2009.

[41] Igor Griva, Stephen G Nash, and Ariela Sofer. *Linear and nonlinear optimization*. Siam, 2009.

[42] Gaël Varoquaux. *Mathematical optimization: finding minima of functions*. Scipy Lecture Notes. 2014. URL: "http://scipy-lectures.github.io/advanced/mathematical_optimization/" (visited on 03/21/2014).

[43] Julien Legriel. "Multi-criteria optimization and its application to multi-processor embedded systems." PhD thesis. PhD thesis, Grenoble, University Joseph Fourier, 2011.

[44] Markos Papageorgiou. *Optimierung: statische, dynamische, stochastische Verfahren*. Springer DE, 2012.

[45] SJ Wright and J Nocedal. *Numerical optimization*. Vol. 2. Springer New York, 1999.

[46] Jorge Nocedal and Ya-xiang Yuan. "Analysis of a self-scaling quasi-Newton method." In: *Mathematical Programming* 61.1-3 (1993), pp. 19–37.

[47] Özgür Turhan. *Optimizasyon Yöntemleri*. Turkish. University lecture notes. Istanbul Technical University. Feb. 2010.

[48] Paul T Boggs and Jon W Tolle. "Sequential quadratic programming." In: *Acta numerica* 4 (1995), pp. 1–51.

[49] Moritz Werling et al. "Optimal trajectory generation for dynamic street scenarios in a Frenet frame." In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 987–993.

[50] Kamal Kant and Steven W Zucker. "Toward efficient trajectory planning: The path-velocity decomposition." In: *The International Journal of Robotics Research* 5.3 (1986), pp. 72–89.

[51] Philipp Bender, Julius Ziegler, and Christoph Stiller. "Lanelets: Efficient map representation for autonomous driving." In: *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*. IEEE. 2014, pp. 420–425.

[52] Marcus A Brubaker, Andreas Geiger, and Raquel Urtasun. "Lost! leveraging the crowd for probabilistic visual self-localization." In: *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE. 2013, pp. 3057–3064.

[53] Dirk Stöcker and Immanuel Scholz. *Java OpenStreetMap Editor*. Program. Available online. 2014. URL: `https://josm.openstreetmap.de/` (visited on 03/19/2014).

[54] John Parr Snyder. *Map projections–A working manual*. 1395. USGPO, 1987.

[55] Ruben E. Perez, Peter W. Jansen, and Joaquim R. R. A. Martins. "pyOpt: A Python-Based Object-Oriented Framework for Nonlinear Constrained Optimization." In: *Structures and Multidisciplinary Optimization* 45.1 (2012), pp. 101–118. DOI: 10.1007/s00158-011-0666-3.

[56] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. 2001. URL: `"http://www.scipy.org/"` (visited on 03/26/2014).

[57] Steven G Johnson. *The NLopt nonlinear-optimization package*. Optimization library. Available online. 2014. URL: `http://ab-initio.mit.edu/wiki/index.php/NLopt` (visited on 04/16/2014).

[58] Dieter Kraft. *A software package for sequential quadratic programming*. DFVLR Obersfaffenhofen, Germany, 1988.

[59] ShihPing Han. "A globally convergent method for nonlinear programming." In: *Journal of optimization theory and applications* 22.3 (1977), pp. 297–309.

[60] Michael JD Powell. "A fast algorithm for nonlinearly constrained optimization calculations." In: *Numerical analysis*. Springer, 1978, pp. 144–157.

[61] Charles L Lawson and Richard J Hanson. *Solving least squares problems*. Vol. 161. SIAM, 1974.

[62] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. "The NumPy array: a structure for efficient numerical computation." In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30.

[63] Andreas Brodtkorb. *Numerical Differentiation Tool for Python.* Tool. 2014. URL: `https://code.google.com/p/numdifftools/` (visited on 05/07/2014).

[64] Krister Svanberg. "A class of globally convergent optimization methods based on conservative convex separable approximations." In: *SIAM journal on optimization* 12.2 (2002), pp. 555–573.

[65] Michael JD Powell. "A direct search optimization method that models the objective and constraint functions by linear interpolation." In: *Advances in optimization and numerical analysis.* Springer, 1994, pp. 51–67.