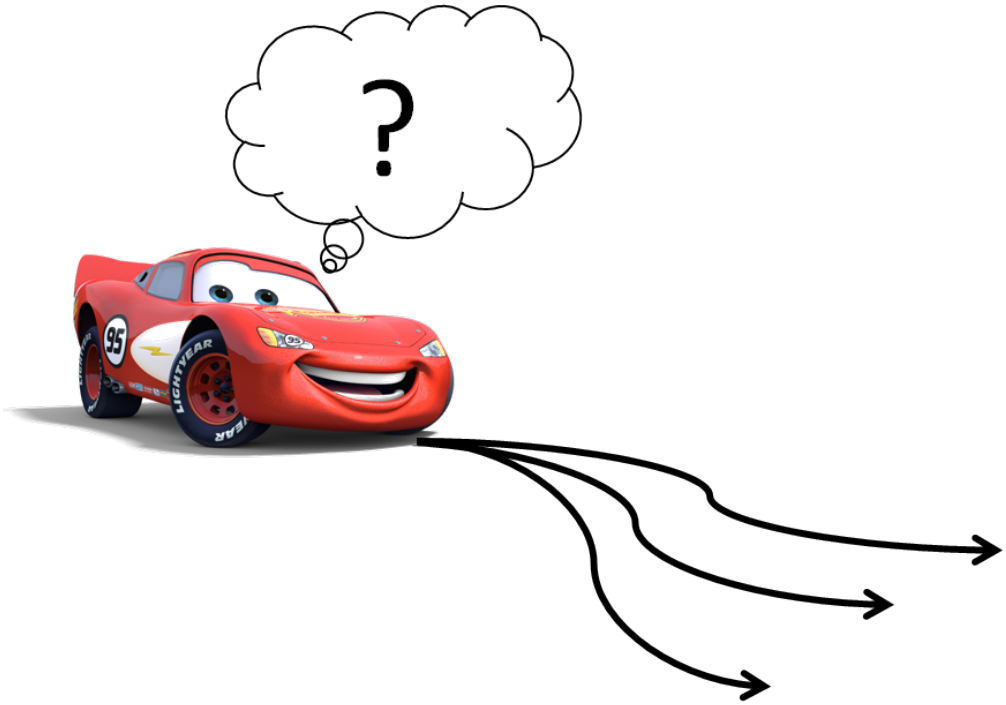
CHALMERS



Path Planning for Highly Automated Vehicles

Master's Thesis in Systems, Control and Mechatronics

Robert Hult
Reza Sadeghi Tabar

Department of Signals and Systems
Automatic control, Automation and Mechatronics
Chalmers University of Technology
Gothenburg, Sweden 2013
Master's Thesis 2013:1

Abstract

The vision of highly automated vehicle systems has for the last two decades enticed researchers in both academia and industry. The work in this thesis focuses on the *path planning* subset of the field, which addresses the problem of how a vehicle should navigate through an obstacle filled environment. The purpose is to study and evaluate the state of the art of path planning approaches and propose a solution that is feasible under the resource constraints present in today's passenger vehicles and that satisfies demands on robustness and passenger comfort. Due to the complexity of the general path planning problem, the scope is limited to operation on highways and country roads, including entries and exits through turns.A modular solution is presented where the planning is reduced to selection among a discrete set of simple trajectories. The selection is based on a filtered compound cost function that capture various performance aspects, e.g. proximity to other vehicles and comfort. For both highway and turning manoeuvres, the simple trajectories are found by solving a constrained optimal control problem subjected to simplified representation of the vehicle dynamics. It is shown that the trajectories for the highway case takes the form of a polynomial when a particle model is used to approximate dynamics together with a specific cost functional. For the turning case however, it is shown that the corresponding constrained optimal control problem is of numerical nature and the reformulation to a non linear program by assuming a parametrized form of the control is detailed. The importance of the initial guess to the numerical solver is demonstrated and a novel way of generating these algorithmic starting points using function fitting via Feed Forward Neural Networks (FFNN) is presented. The benefit of using the highly accurate FFNN based initial guess is demonstrated, and its superiority over other methods presented in the literature is shown. Results from extensive simulations are presented which demonstrates that the proposed system is capable to solve planning through complex traffic scenarios. The model simplifications made are justified by a comparison between the dynamics of the simple model and a non linear bicycle model, where the latter is forced to follow the path of the former. The resulting Differential Algebraic Equations are solved through Dymola and used to determine under what conditions the simplification is valid. The thesis is concluded with a discussion on the results touching on both the benefits and disadvantages of using a modular approach to the path planning problem.

Acknowledgements

We performed the work presented in this thesis during the spring 2013 with close connections both to Chalmers and Volvo Car Corporation. It has been an interesting journey with many surprises along the road, and without the help of a number of persons it is likely that it would have been a lot less enjoyable. First of all, our supervisor Dr. Stefan Solyom at Volvo Car Corporation deserves special appreciation for his interest, guidance and critical questioning along the way, without his ideas the result would very likely be something completely different. A second acknowledgement goes to our examiner and advisor at Chalmers, Professor Jonas Sjöberg who similarly contributed with insights both on the actual work and the presentation of the results. In no particular order we would also like to thank Dr. Gabriel de Campos, Dr. Nicolce Murgoviski, Associate Professor Torsten Wik as well as Professor Claes Breitholtz, all at the department of Signals and Systems, for listening to our questions around optimal control, as well as Professor Bengt Jacobsson at the Vehicle Dynamics Group of the Applied Mechanics department for private tutoring in vehicle modelling. We would also like to thank Olaf Verstrijden from TU Eindhoven for rewarding discussions in the initial phase of the work. Last, but not least, a great thanks goes to our friends and families who have shown great patience and understanding during the course of the project.

Robert and Reza, Göteborg 20/6/13

Contents

1

Introduction

Today's ground vehicular systems are moving more and more towards autonomy. Advanced driver assistance systems (ADAS) have excited immense interest in academia and within the automotive industry. Recent advancement in sensory systems and continuous reduction of computational costs have led to a significant escalation in number of such functions available today. Functions such as adaptive cruise control, lane keeping aid, parking assist, traffic jam assist and collision avoidance systems are features in vehicles already in production. At the same time concept of intelligent transport systems (ITS) is rapidly gaining popularity. Standardization of vehicle to vehicle and vehicle to infrastructure communication protocols is paving the way for the next generation of ITS solutions. Achieving higher levels of safety, improved comfort, lower fuel consumption and decreased road congestions are the incentives that fuel the research in this area.

On a research level, several projects focused on autonomous vehicles have existed with perhaps the most prominent being the Defense Advanced Research Projects Agency (DARPA) challenges. On an industrial level on the other hand, such examples are scarce. However, with the technical prerequisites becoming available and as the general public becomes more receptive, the vision of highly autonomous vehicles gets closer to realization. Considerable effort is being directed towards autonomously navigating vehicles in different urban environments. This calls for enhanced sensor systems and environment representation, decision making and path planning solutions as well as control algorithms.

1.1 Purpose

The question of how to generate an optimal trajectory for an autonomous vehicle to follow is one of the challenges in this area. Focusing on the path planning problem, the presented work aims to study and evaluate the state of the art of path planning approaches and propose feasible solutions considering the current capacity of the vehicles. The proposed solution shall fulfil the following criteria:

- **Feasibility of implementation given the processing power available in passenger vehicles**

  To emphasize the importance of having a computationally cheap solution, it can be noted that in the electronic architecture of a vehicle in production today, allotted memory to a comparable function is typically in the kilobyte range while the designated processor operating in sub-gigahertz frequencies is shared with a multitude of other functions. This is a major difference between an industrially feasible solution compared to those used in research experiments (see figure 1.1).

- **Robustness and guaranteeing kinematic feasibility**

  The path planner shall only generate paths which are traversable by the vehicle. In other words, kinematic constraints of the vehicle should be taken into account.

- **Ease of integration with the currently available sensors typically seen in passenger vehicles**

  Radars and cameras which are normally present in today's passenger vehicles provide information about the surrounding objects and road references. The path planner shall not require any extra sources of information than the aforementioned.

- **Account for passenger comfort**

  Passenger comfort is considered one of the prime objectives of the path planning system in the automotive context. From a marketing point of view, failing to satisfy the customer's comfort criterion will render such a function undesirable.

The first phase of this work comprises a literature study for the purpose of assessing the potential of the available solutions given the demanded criteria. With this knowledge acquired, the next phase consists of development of a modular solution for driving in a structured environment, specifically for highway driving and turning. Next a series of simulations evaluate the performance of the proposed solutions.

(a) (b)

Figure 1.1: 1.1(a) University of Carnegie Mellon's entry to the DARPA Urban Challenge. 1.1(b) Volvo V40 with a sample electronic control unit. Such unit typically hosts several functions running on sub-gigahertz processor and with allotted memory in the kilobyte range.

1.2 Scope

This project assumes the presence of a sensing and perception layer which provides a representation of the environment throughout the planning horizon. In other words, object tracking and obstacle detection are not in the scope of this work. Likewise, development of a path following controller to follow the generated trajectory is not considered. However, being conducted in cooperation with Volvo Car Corporation, this work gives special attention to the software architecture and the currently available hardware and software modules specific to this company. The solution is intended for normal non-emergency driving conditions on highways and rural roads including entering and exiting scenarios. In other words, evasive manoeuvres and off-road driving are not considered.

Per request from Volvo Cars the longitudinal velocity of the vehicle is not to be included in the planning and is therefore neglected in this subsystem. Instead, the well tested Adaptive Cruise Control (ACC) module already in production is to be used. This controller adjusts longitudinal velocity by means of feedback from the absolute velocity of the host as well as from the distance and relative velocity with respect to a target vehicle. The behaviour of the ACC can therefore be said to operate in two modes: one where the host strives to maintain some set speed, the other where it instead adapts its speed to that of a preceding target whilst keeping some dynamically varying reference distance. As a result, the degrees of freedom available are restricted to the lateral direction and will basically result in reference trajectories for a steering controller.

Beside this uncertainty, the restriction of the planners authority stemming from the use of the built in *ACC* already limits the scope over which planning can be made, and greatly reduces the planning ability. The following merging scenario not uncommon on

highways, can be considered an illustrative example of the latter. Without claiming to cover all possible situations, this implies that a restriction in the planning dimensions yields suboptimal results.

Example: The host vehicle (H) is approaching a slower moving vehicle (T) from behind. In the adjacent lane yet another vehicle (A) is travelling approximately at the same speed as the host alongside it, as illustrated in Figure 1.2. Almost regardless of what criterion that now is chosen for performance evaluation, a multitude of options arises. For example, the host can accelerate over some distance and then change lane, trajectory 1 in the figure. Alternatively it can pre-emptively slow down and thereafter change lane, trajectory 2 in the figure. Removing speed and longitudinal positioning however, the ACC essentially leaves one option open, namely to approach (T) at the present speed, slow down relatively close to achieve the set distance, await the passing of (A), *then* change lane. Needless to say, this might be far from the optimal way of performing such a manoeuvre.

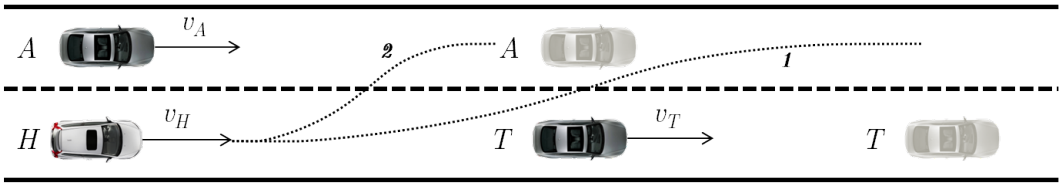


Figure 1.2: Obstructed Lane Change/Merging scenario. Here $v_H \approx v_A$ and $v_H > v_T$. The lighter shaped vehicles represent positions after some time has passed. Note that the image is meant to be schematic and that the distances not are to be interpreted as physical.

1.3 Organization of the Report

The next chapter of this report provides some background information, introduces the path planning problem in its general form and formulates the problem in the automotive context. Chapter 3 elaborates the development of a modular solution for *(i)* highway driving and *(ii)* turning. For the first scenario, derivation of an optimal path for maximizing comfort and satisfying given constraints are explained along with a model for representing moving obstacles. The next part proposes a solution for optimal trajectory generation under kinematic constraints for a turning scenario. Evaluation of the resulting trajectories along with simulation results and performance analysis of the proposed methods are presented in Chapter 4.

2

Background

This chapter first introduces the path planning problem in its most general case and gives a short overview of common approaches taken as well as some history of the field. The general path planning problem is formulated first as an optimal control problem and then the computer science take on the problem is explained. After this, the path planning problem in an automotive context is described, highlighting the important differences with respect to the general problem. The chapter then concludes with some remarks on the commonly used planning methods and their favourability in comparison with the proposed method in this work.

2.1 Theoretical Preliminaries and Definitions

Definition 1 *A path is in this context defined as a sequence of interconnected nodes in a graph representation of* $\mathbb{R}^2$

Definition 2 *A trajectory is a continuous curve connecting two points in* $\mathbb{R}^n$

The two terms *Trajectory* and *Path* are defined in different ways by many authors while some use them interchangeably. The distinction made in this thesis is that a trajectory *always* is a continuous curve in some space, possibly the state space of some system, while a path can be a discrete sequence. Since a physical system only can execute, or follow, continuous paths the latter commonly takes the form of a discrete sequence of *trajectories*.

Definition 3 *Configuration space or C-space of a robot refers to a set of all possible positions and orientations of the robot.*

Definition 4 *A nonholonimic mechanical system is a system under velocity constraints, the derivatives of which do not integrate to the position constraints. Such constraints are referred to as nonholonomic constraints.*

In mobile robotics for instance, nonholonomicity is typically signified by the *rolling without slipping* property. Parallel parking of a car, as depicted in figure 2.1 exemplifies the effect of nonholonimic constraints.

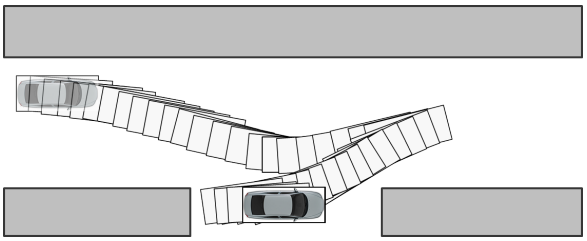


Figure 2.1: parallel parking of a car

In mathematical terms, a kinematic equality constraint can be defined as

$$\begin{aligned} &F(q,\dot{q},t) = 0 \\ &q \in A \\ &\dot{q} \in B \end{aligned} \tag{2.1}$$

with q being a configuration in the *configuration space* A of a robot and B the tangent space of A at q. The equality constraint2.1 is nonholonomic if it cannot be integrated into the form $F(q,t) = 0$. Assumption of linearity of F with respect to $\dot{q}$ will lead to the above-mentioned *rolling without slipping* property in a car. On the other hand, a nonholonomic kinematic inequality constraint of the form $F(q,\dot{q},t) \leq 0$ corresponds to e.g. bounds on the steering angle of a car [1].

2.2 General Path Planning Problem

The path planning, or motion planning problem, has long been seen as one of the fundamental problems in robotics. Originally arising from the need for robots to find collision free paths between two locations in an obstacle cluttered environment, it has expanded and continues to attract interest from both research and academia. Even though the path planning problem is different for manipulators and mobile units much of the research done can be applied to both fields. This introduction will however have a slight

focus on the mobile case since it gives some background to the automotive problem presented in the next section.Upon finding references to classes of algorithms the reader is encouraged to look for the details as given in Appendix A

2.2.1 General Problem Formulation

For all applications, the general path planning problem can formally be described as follows:

Problem 1 *"Given an arena, or environment A, containing obstacles O find a path p in $A \setminus O$ such that p connects the points a,$b \in A \setminus O$ minimizing the measure M".*

Here, A defines the boundaries of the problem, i.e. the total search space. The path endpoints a and b are in the free part of A defined by $A \setminus O$. M can be any number of measures, including functions of dynamic states of the agent that is to traverse the path or the effort demanded, but is commonly directly related to properties of the path itself, e.g. length, curvature etc. The graphical representation of the problem is given in Figure 2.2.

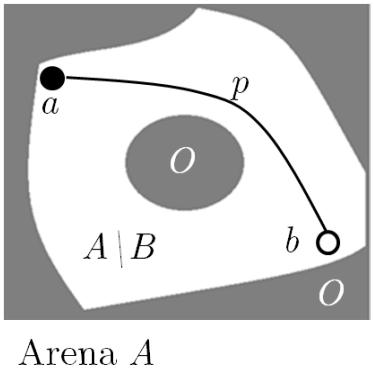


Figure 2.2: Visualization of the definition given in Problem 1. The white area is the free parts of the arena whereas the gray represents the parts that are not traversable.

2.2.2 Control Theoretic Approach

By examining Problem 2.2 from a control theoretic perspective, one can see the nature of the problem resembles that of an *optimal control* problem. In this class of problems, the objective is to find the control, and subsequently controlled trajectories, taking a dynamical system from one state to another, respecting constraints imposed both on the

control action and the states themselves. In the robotics perspective, the latter can then be formulated such that both workspace obstacles restricting the pose and performance measures restricting speeds, accelerations etc. are included, thereby presenting a framework in which the path planning problem can be approached. In the most general case the optimal control formulation of the path planning problem can then be stated as

$$\begin{aligned} \min_{u(t)} \quad & \int_0^{t_f} L(x(t),u(t))dt + \phi(x(tf),u(tf)) \\ \text{s.t.} \quad & \dot{x}(t) = f(x(t),u(t)) \\ & x(0) = x_0 \\ & \psi(x(t_f)) = 0 \\ & x(t) \in X \\ & u(t) \in U \end{aligned} \tag{2.2}$$

Here, $x(t) \in \mathbb{R}^n$ is the states of the system, $u(t) \in \mathbb{R}^m$ is the control signals, t_f is the possibly free final time, x_0 is the start state, $\psi(x(t_f))$ the conditions related to the final state, $f(x(t),u(t)) : \mathbb{R}^{m+n} \mapsto \mathbb{R}^n$ the system dynamics. $L(x(t),u(t)) : \mathbb{R}^{m+n} \mapsto \mathbb{R}$ is the path cost to go while $\phi(t_f)$ is the final cost.

2.2.3 Computer Science Approach

Much of the work in path planning was done primarily within the fields of Artificial Intelligence and Computer Science which have put their mark on the approaches that have become prominent. Even though there are important exceptions, especially some put forward in the last decade, kinematics and dynamics are largely ignored [2]. Looking back at Problem 2.2, this means that the dynamics of the system are neglected. The solution to the original path planning problem is then usually found approximately by iterative search schemes, similar to other problem solving techniques within the field. The most common approach is to first discretize the obstacle free part of A (c.f Problem 1), either by decomposing it into cells or to skeletonize it into nodes with connecting edges. A schematic illustration is given in figure 2.3

Relationships between the discrete elements are then defined, so that for each node or cell there exists a set of neighbouring elements. For each of these a cost of transition is derived from the quantity that is to be minimized over the complete path, representing the partial cost of moving from the first element to one of its neighbours. The discretization and connections are defined such that no nodes or connections passes through any obstacle, thereby letting all transitions on the discretized space map to collision free trajectories in the original workspace. Having done this, the problem can then be thought of as a bidirectional graph or tree originating in the start point, were the discrete elements are nodes and the connections to the neighbours are the edges. Following the principles of

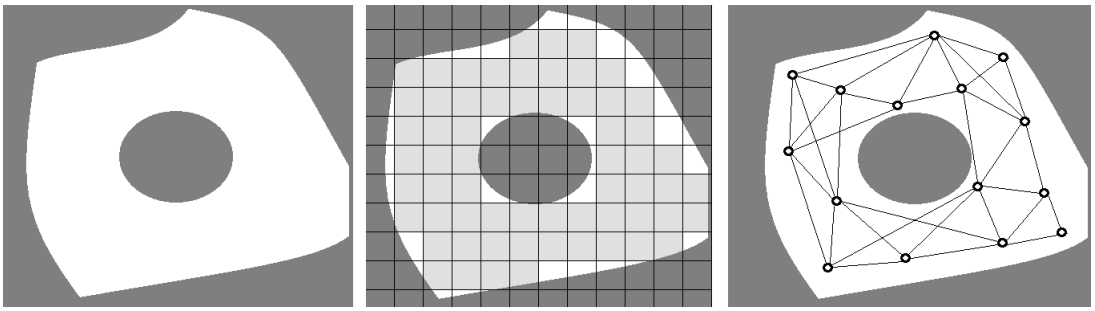


Figure 2.3: Schematic demonstration of common dsicretization schemes. **Left:** The original search space **Center:** Search space decomposed into cells **Right:** Search space skeletonized

dynamic programming the solution is then taken as the path in the graph having the lowest total cost from the start node to the endpoint, found by some graph searching algorithm or general dynamic programming scheme. The focus of the research within this approach has therefore to a large extent been to improve or reform the techniques used for discretization and to increase the efficiency of the search algorithms themselves. Some details on the present state-of-the art are given in Appendix A.

As mentioned earlier, computer science approaches typically ignore the involved kinematics and dynamics. This gives limited or no guarantees on the feasibility of the solution and no possibility to keep certain dynamic quantities within any pre described bounds. Further, most methods proposed use little or no a-priori knowledge of the solution and do not assume a specific structure of the problem, focusing instead on versatility and the ability to function well in a multitude of environments. Even though it is advantageous in many applications, notably free robot navigation in complex environment, this usually necessitates an exploration of a large part of the space of possible solutions, in general leading to high computational loads.

2.2.4 Combining Computer Science and Optimal Control Approaches

As noted by among others Lavalle [2] recent developments have spurred an increased interest in both the AI and control communities for the approach taken by the other, and a common ground have been established. During the last fifteen years, a number of methods based on skeletonization and search have been introduced, where the connections between the nodes in principle are found by solving smaller restricted optimal control problems. By this, guarantees and limitations regarding system dynamics can be stated together with some measure of algorithmic completeness and bounds on suboptimality [4],[5],[6]. There are of course other approaches bridging the gap between the fields, but the one mentioned presents a good example of hybridization between the two originally different approaches.

There are presently a large number of applications, both using optimal control related algorithms, search based ones, combinations of both or others still. Since the number of mobile robots has exploded and now is more numerous than their stationary counterparts the field has increasingly become more important. This is especially true since they, as in the application related to the central topic of this thesis, takes a larger part of everyday life. Examples of areas where some of these algorithms have been used are shown in figure 2.4.

Figure 2.4: Users of Path Planners: **Left:** NASA Rover *Curiosity*, **Center:** Home utility robots, **Right:** US Military UAV *MQ-9 Reaper*

2.3 Automotive Path Planning

The interest in automation of personal vehicles has been growing successively the last 20 years. In industry the primary focus has been on functions for active safety and driver support such as Adaptive Cruise Control (ACC) amd Lane Keeping Aid (LKA), all sharing a relatively low level of autonomy, never keeping the driver completely out of the loop. Recent developments have however sought to increase the level of autonomy and have initially led to the introduction of semi or fully autonomous functions, most recently for parallel parking. There is a desire in industry to advance this field further as it is believed that increased autonomy, and finally complete autonomy, will take on an important role in the commercial road vehicles of the near future. Evidence of this effort can be seen in projects with industrial participation like *SARTRE* and *GCDC*. In contrast, the technological frontier seen in more academic projects lies far ahead of that present in industry, and has demonstrated fully autonomous behaviour on several platforms over the last decade. Notable applications include those of the participating teams in the *DARPA Grand Challenge (DGC)(2005)* and *DARPA Urban Challenge (DUC) (2007)* and lately the *Google Car*, performing fully autonomous operation both in enclosed areas and on public roads. In terms of path generation these applications commonly incorporate algorithms with a focus on actual *planning*, i.e. in line with the example above, they adapt to a situation well before it is reached to achieve some global optimality property. This differs significantly from common industrial practice, e.g. that found in automatic emergency manoeuvres. In general, these act on a problem when it

is encountered and are thereby is more *reactive* in nature whereby they commonly fail to achieve the same measure of optimality. The efforts made both in industry and academia towards full autonomy have clearly shown the need for an adaptation of the existing path planning algorithms to an automotive context, and have placed the problem as one of the important fields of study together with *Decision Making*, *Sensing* and *Control*.

2.3.1 Differences to path planning in robotics

Automotive path planning encompasses considerations regarding the involved human factors such as passenger comfort and safety measures. Accounting for these considerations is what differentiates the automotive path planning from similar problems in the field of robotics. These human factors, like in other driver support systems, play an important role in user acceptance and subsequently commercial success or failure of the end product.

These considerations raise the need for explicit handling of the dynamics of the controlled vehicle. In terms of safety, there is a need to guarantee that a computed path is collision-free and kinematically feasible within some imposed restrictions. Demands on ride comfort, in terms of vehicle dynamics is expressed as explicit limitations on development of certain states.

The optimal control formulation of the path planning problem presented in Section 2.2.2, we can show that the above mentioned human factors can be integrated into this formulation. Comfort and safety requirements can be fulfilled by identifying the involved system states and defining the desired constraints on those states. More specifically, in Problem 2.2, X may contain constraints on states x such as acceleration, velocity, etc. to fulfill some safety criteria, as well as guaranteeing a collision-free path. Requirements on passenger comfort can be integrated into the cost function to be minimized.

As a remark on the issue of passenger comfort, it is commonly believed that high levels of jerk (time derivative of acceleration) and acceleration are the prime contributors to ride discomfort, see for example [7] and [8]. Both quantities can be directly perceived by the user, as opposed to velocity or position, and is believed to impact the perception of vehicle movements and therefore the experienced ride quality. Despite the importance of these effects thorough experimental studies are scarce and numerics for limiting values are few.

2.3.2 Scenarios for autonomous drive

Much of the research done on systems for autonomous drive divides the scenarios into three main categories; *Off-road Operation, Operation in Zones* and *Operation in structured environments*, differentiated by how much can be assumed about the environment

[10],[11],[12], see Figure 2.5 for a schematic example. *Operation in Zones* typically en-

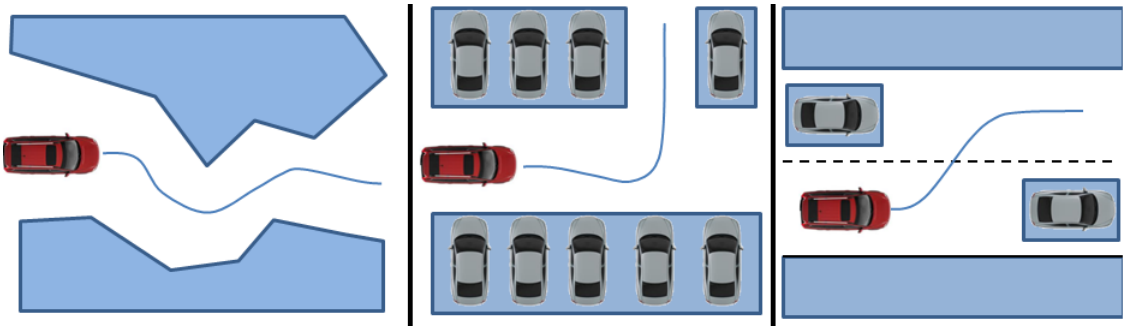


Figure 2.5: Schematic Illustration of the different autonomous driving scenarios.**Left:** Off-road driving. No assumptions can be made regarding the structure of the scenario, thereby increasing the problem complexity. **Center:** Zone driving. End of trajectory contains demands on all states, including vehicle heading. Assumptions are made on the geometry of obstacles, and the flatness of the ground which simplifies the problem. **Right:** Structured Driving. End of trajectory concerns only certain states, e.g. lateral position from road center, velocity and acceleration. Directional information implicitly given by the scenario structure.

compasses parking lot scenarios, driving among pedestrians and other low speed situations. As these can differ significantly and many manoeuvres needed can demand operation at the kinematic performance limits of the vehicle, e.g. regular parking with maximum steering angle, several research works have proposed exhaustive search schemes incorporating detailed models of the vehicle kinematics, commonly using both AI related algorithms and optimal control solvers [13]. Unlike this category, planners for *Operation in structured environments*, e.g. road bound operation at higher speeds, rarely need to come close the kinematic limits of the vehicle, except for the case when emergency manoeuvre are performed. The road itself and the direction of travel offer structure that limit the set of desirable paths and thereby the space in which to search for optimal ones. Indeed, this structure has frequently been used in various research projects, for example [14],[15], often leading to significantly simpler algorithms than for its zone navigation counterpart. Yet another difference is the differences in path endpoints. Where the zone navigation often demands precise positioning and an explicit goal state this is rarely the case for structured driving on roads. Instead the lane in which the vehicle travels and the offset to a lane center are usually only considered because the heading and position along the road are less relevant. Due to its similarities with ”regular” robotics path planning (no specific environment structure, lower speeds etc.) and its small relevance to this thesis, the *Off-road Operation* category will not be described further.

2.4 Choosing a Feasible Approach

At the initial stage of this work, in the quest for selecting a path planning approach to satisfy all the criteria as described in 1, a selection of commonly used methods were evaluated. A summery of these approaches can be found in Appendix A. It can be noted that the majority of the summarized approaches fail to fully satisfy the criteria described in the Section 1.1. Optimal control based solutions tend to have high computational loads even on dedicated processors. Graph-search methods cannot guarantee a kinematically feasible path on their own. Algorithms based on randomized sampling have the capability of using advanced kinodynamic motion models and generate paths for complex manoeuvres, but they prove to have high demands on computational resources. Limited or no possibility to include kinematics in the geometry based approaches make them less of a favourable solution, and the problem of local minima in the potential field approach does not satisfy the robustness criterion. In the state-lattice method, both generation and storing large number of trajectories are beyond the intended processor and memory capacity for this work.

Except for the computational burden, shortcomings of these algorithms can be addressed by using a more general hybrid solution comprising of a selection of these methods. The most note-worthy of real-world implementations of the more general solutions to the path-planning problem for mobile robots can be those used in the DARPA challenges. The participants normally utilize hybrid solutions combining a selection of the discussed classes of algorithms to form a general and adaptable solution. These solutions are typically closely coupled with an elaborate perception module hosting a magnitude of advanced sensory equipment. On the computational side, the real-time platforms are typically tailored to suit the high demands of the implemented algorithms.

From the perspective of today's automotive industry, integration of such perception systems are far from realization due to high costs and factors such as design and customer appeal considering placement of sensors. Most importantly, the computational costs are not justifiable in a mass-produced vehicle. Moreover, approval tests and procedures for experimental validation of a general path planning solution for automotive applications will be costly and complex. Hence, the risk of a compromise in safety may increase. Due to their highly integrated nature, maintainability and scalability of such a solution will also be challenging in an industrial setting.

The current driver support systems mainly consist of a number of specific modules optimized for a particular application. Adaptive cruise control, lane keeping aid, assisted parking, and prototypes in vehicle platooning are such functions. In the context of path planning, these functions follow the approach of path planning for structured environments as described in Appendix A.7. This modular design provides the possibility of effective verification and is easier to maintain and scale.

With this in mind, the proposed method as described in the next chapter takes on a

modular approach. Feasibility of integration into a vehicle in production today has been of particular importance in this work. The provided solutions aim to have no significant demand on the sensory perception modules, nor on the designated processors already present in the vehicle.

3

Proposed Planning System

This chapter contains a detailed description of a path planning system for automotive applications. Drawing from the results of the previous chapter comprehensive algorithms capable of handling a multitude of environments and situations have been deemed inappropriate since they, in general, are too complex for on-line implementation in the currently available vehicles. Instead, the proposed planning system is situation specific, with different algorithms for different scenarios, which reduces overall complexity and thereby allows rapid computation. The focus of the planner is operation on highways and country roads and is subdivided into two main categories: *Highway Operations*, e.g. lane changes, overtakes and small lateral movements within a lane and *Turning*, e.g. leaving a country road by means of a right or left turn. The main idea is to approximate the original Optimal Control formulation from Problem 2.2 by a selection from a set of tentative trajectories. A framework is presented for the rating and selection between the plausible alternatives. The trajectories for highway manoeuvres are shown to have a very simple analytic form under certain simplifications while the counterparts for turning are retrieved through the solution of a non-linear optimization problem. The latter is troubled with numerous local minima and a novel method for initiation of the optimization algorithm based on Feed Forward Neural Networks is presented, which mitigates the issue.

3.1 System Overview

The path planning problem as presented in Problem 2.2 and the common approaches taken to solve it was discussed in the previous chapter. It was demonstrated that most algorithms were coupled with different, but severe drawbacks ranging from complexity

issues to the inability to account for system dynamics. Drawing from these experiences, the system proposed in this Chapter aims to tackle the inherent difficulties by utilizing the restrictions of the project to form a specialized approximation to original problem, which for convenience is restated below

$$\begin{aligned} \min_{\mathbf{u}(t)} \quad & J = \int_0^{t_f} L(\mathbf{x}(t),\mathbf{u}(t))dt + \phi(\mathbf{x}(t_f),\mathbf{u}(t_f)) \\ \text{s.t.} \quad & \dot{\mathbf{x}}(t) = f(\mathbf{x}(t),\mathbf{u}(t)) \\ & \mathbf{x}(0) = \mathbf{x}_0 \\ & \psi(\mathbf{x}(t_f)) = 0 \\ & \mathbf{x}(t) \in X(t) \\ & \mathbf{u}(t) \in U \end{aligned} \tag{3.1}$$

In this formulation, the allowed state space set $X(t)$ is intended to be read as covering all the important aspects, e.g. obstacle avoidance and limitations on acceleration and velocity. Note that the time varying of nature of nature of X is written out explicitly, emphasizing the need for adaptation to dynamically moving obstacles and other environmental changes. The final time t_f is considered fixed and the problem is solved over a finite horizon. It is again stressed that the problem statement given in Equations 2.2 and 3.1 merely are symbolical in nature and only intended to capture the conceptual width of the problem. As noted in the previous chapter, a wide variety of *general* approaches has been proposed to approximate the solution to Equation 3.1, all coupled with severe drawbacks when commercial automotive applications are intended. Striving to achieve a solution suitable for implementation on the hardware platforms available today that still includes a model of vehicle dynamics with constraints it is evident that a number of simplifications has to be made.

Approximating the continuous optimization problem presented in Equation 3.1 with a discrete counterpart the method proposed in this thesis is similar in spirit to the work presented in [16],[12] and [17]. Conceptually, the method also share ideas with the *State Lattice* methods discussed in Appendix A.6. At each time instant, i.e. sample, a set of simple trajectories derived from the constrained vehicle dynamics from Equation 3.1 are generated from the current position to a number of tentative targets. These trajectories represent possible ways in which the host vehicle can change its position and ways in which this is performed (i.e. its corresponding state evolution) over the course of some short time period into the future. Simultaneously the commonly highly dynamic environment is simulated, i.e. the behaviour of other road participants is predicted, based on the currently available information. Following this, the predicted environment is used to decide if the tentative trajectories of the host vehicle are feasible (i.e. does not lead to a collision, stays on the road etc.). Those that fulfils these requirements are then attributed a score after their fitness in terms of some selection criterion. At this point, each trajectory with a score is thus feasible in terms of the constraints provided

in Equation 3.1. By designing the score after the objective J in the same equation, an approximation to the problem is found by selecting the best of these feasible trajectories.

The following section further details how the method approximates the problem presented in Equation 3.1 and describes its operation through an illustrative example. For clarity, the restrictions of the project to road bound operation is again emphasized with the additional constraint that the proposed system only handles manoeuvres on larger roads and turns.

3.1.1 Conceptual Main Steps

Consider the problem of overtaking a slower moving vehicle on a highway and the generation of the optimal trajectory using the solution to Equation 3.1. The time horizon is assumed to be long enough to allow planning of the complete manoeuvre, i.e. initiating, performing and finishing the overtake, without violating any of the constraints. Further, the objective J is assumed to be carefully designed to capture the different aspects presented in 1.1 and a high quality dynamical model $f(\mathbf{x}(t),\mathbf{u}(t))$ is assumed. By this the optimal trajectory is the safest, and most comfortable one that can be found. A schematic representation of the scenario is given in Figure 3.1.

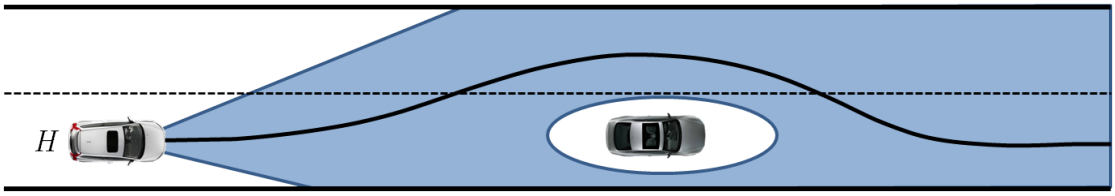


Figure 3.1: Visualization of the overtake scenario. The blue field represent the physical space allowed by the state constraints $\mathbf{x}(t) \in X$ from Equation 3.1. The white areas similarly corresponds to disallowed regions. The black curve is intended to show the optimal trajectory according to some objective J.

The first restriction is made by shortening the horizon to the extent of the available sensors. The current vision-radar combination gives readings up to 150 meters, which for highway speeds between 70 and 90 km/h is covered in roughly 7.5 to 6 s. The highly dynamical environment reduces the benefit of planning beyond this range and essentially makes all calculations involving predictions of other road participants over longer horizons inaccurate. Note however, the implication that very few manoeuvres, e.g. lane changes, overtakes etc. will be able to complete within the allotted time window, and that the planner therefore almost always will be partial in nature. The effect of the restriction on the studied example is shown in Figure 3.2, where the set of feasible solutions is reduced and the planning of a complete overtake therefore no longer is possible.

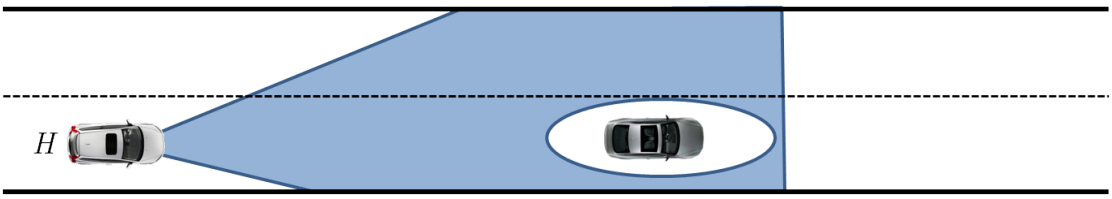


Figure 3.2: Visualization of a reduction in the planning horizon. Using the descriptions from Figure 3.1, the allowed set now fails to encompass the physical space necessary to complete the manoeuvre.

The first step of the algorithm is to generate a set of simpler trajectories that satisfies the constraints in Equation 3.1. Formally, these are retrieved as the solution to

$$\begin{aligned} \min_{\mathbf{u}(t)} \quad & M = \int_0^{t_f} \hat{L}(\mathbf{x}(t),\mathbf{u}(t))dt \\ \text{s.t.} \quad & \dot{\mathbf{x}}(t) = \hat{f}(\mathbf{x}(t),\mathbf{u}(t)) \\ & \mathbf{x}(0) = \mathbf{x}_0 \\ & \mathbf{x}(t_f) = \mathbf{x}_f \\ & \mathbf{x}(t) \in \hat{X} \\ & \mathbf{u}(t) \in U \end{aligned} \tag{3.2}$$

The main point here is to save computation by replacing the dynamics f with a simplified counterpart $\hat{f}$, remove the explicit treatment of moving obstacles, i.e. replace $X(t)$ with $\hat{X}$ and explicitly specify the endpoint, i.e. letting $\mathbf{x}(t_f) = \mathbf{x}_f$ as opposed to $\psi(\mathbf{x}(t_f)) = 0$. Note the different objective M formed over the trajectory through $\hat{L}$. The idea is to retain a subset of the original objective J, or more precisely a subset of the properties introduced in 1.1, using a formulation that allows rapid computation. Specific details are refrain from at this point, as they will be detailed for the highway manoeuvre and turning scenarios in 3.2 and 3.3 respectively.

Using this, a set of trajectory endpoints $\mathbf{x}_f$ are defined inside the feasible set of solutions of the original problem in Equation 3.1, and trajectories leading to them are computed. These are now considered tentative solutions to the original problem stated in Equation 3.1, with different quality in their respective approximation of the original solution. A schematic visualization of this step is given in Figure 3.3, where a number of feasible alternative trajectories are drawn.

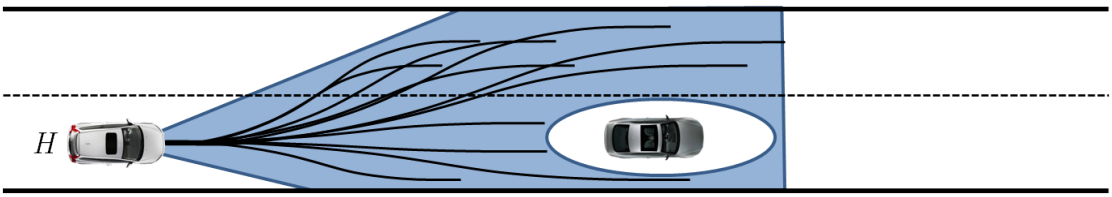


Figure 3.3: Illustration of the generation of feasible trajectories. The blue and white fields are defined according to the caption in Figure 3.1 and the black curves represent the tentative trajectories.

Following the endpoint definitions and subsequent trajectory calculations, the goal is to find the trajectory in the calculated set that gives the best approximation to the solution of Equation 3.1. Using the original objective J and calculating its value over each trajectory, a score can subsequently be assigned to each alternative. Since the original problem in Equation 3.1 is formulated as a minimization, the best approximation is taken as the lowest scoring alternative.

It is stressed that it is in the score attribution that the dynamics of the environment enters the algorithm. Being removed previously in Equation 3.2, moving obstacles are modelled and predicted over the time horizon t_f and the score J computed from the predicted interactions with the host as it executes each trajectory. Consider as an example the proximity to another vehicle, moving at a different speed, which will change in value as the movement of it and the host is simulated. In the special case of a predicted collision, the trajectory is removed completely from the process and discarded.

The scoring and selection process is visualized schematically in Figure 3.4, where the score is represented by the colouring of the tentative trajectories.

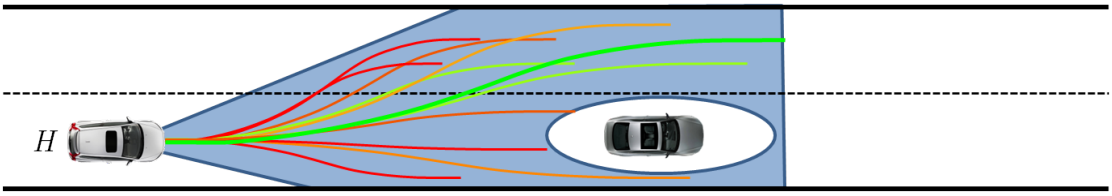


Figure 3.4: Illustration of the score attribution and subsequent selection. The score values are represented with a simple red-to-green color scheme, where red hues correspond to high values of J, i.e. undesirable, and green hues to low values. The trajectory with the lowest value is drawn thicker in bright green. As in previous figures, the blue field represents the feasible, or allowed set

After selection, the best trajectory is executed until the next sample, when the entire process is repeated. The planning is thus done in a receding horizon fashion, much similar to Model Predictive Control. The progression of the algorithm for the overtake example is visualized in Figure 3.5.

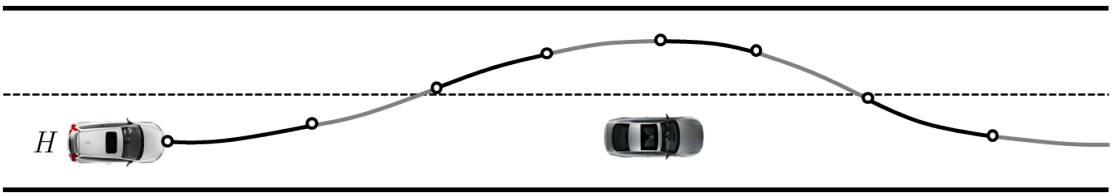


Figure 3.5: Traversed trajectory after several consecutive replanning and execution iterations. The planned and executed trajectories are marked with alternating black and gray solid lines, where each color segment symbolises the result of one iteration. The circular symbols shows where replanning occurs.

Note that the same reasoning is applied to trajectories with endpoints that deviates from the direction of the road, as in exit or entry turns from and to roads and parking lots. A schematic demonstration is given in Figure 3.6.

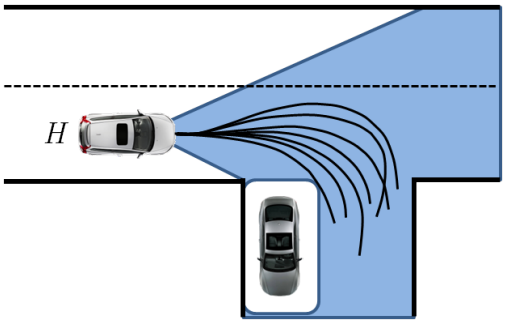


Figure 3.6: Tentative trajectories generated for turning scenarios. Colourings and other markings according to the previous figures.

3.1.2 Algorithm Description

The proposed system can be broken down in to several different modules tasked with subsets of the procedure introduced in the previous section. Naturally, the implementation of the system needs to consider certain practical details due to which additional functionality is included. A graphical overview of the complete algorithm is given in Figure 3.7, where the information flow and internal ordering of the different subsystems is presented. The remainder of this section serves as an introduction to these components.

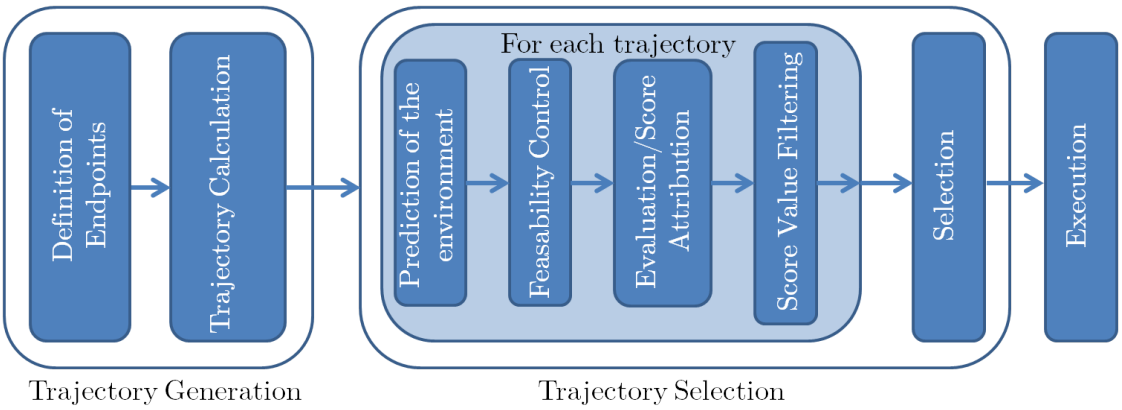


Figure 3.7: Information flow chart for the algorithm presented in the Thesis. Details regarding the purpose of the different blocks are given in 3.1.2.1 and 3.1.2.2.

3.1.2.1 Trajectory Generation

To efficiently utilize as much environmental structure as possible, two different methods for calculating trajectories are used for the *Highway Manoeuvre* and *Turning* Scenarios respectively. The two subsystems will be detailed in 3.2 and 3.3 where motivations and mathematical descriptions are provided.

Definition of Endpoints This component is responsible for providing the targets, or endpoints, of the tentative trajectories, after a predefined rule. Needing only the environment structure as an input, the output is a set of endpoints $\mathbf{x_f}$ that are fed to the trajectory calculation. A description of a endpoint generation rule for the *Highway Manoeuvre* use-case will be given in 3.2.2, where the relative position is retained between the algorithms iterations.

Trajectory Calculation Using the given endpoints a set of trajectories are computed as the solution to a simplified problem, as exemplified through Equation 3.2 in the above example. Consequently, all explicit handling of other road participants are removed and the road considered obstacle free. The output from this component is a collection of points along each trajectory, i.e. a discretized representation of the calculated result. Descriptions on the derivation and calculation of the methods used are given in 3.2 and 3.3 for the *Highway Manoeuvre* and *Turning* scenarios respectively.

3.1.2.2 Trajectory Selection

The selection of the generated trajectories essentially serves two purposes. The first is to ensure that all constraints of Equation 3.1 are respected, specifically those relating to the interaction with other vehicles present in $X(t)$. The second is to approximate the minimization in the same equation by providing an approximation of the objective J over each calculated trajectory. As indicated in Figure 3.7, this is done in several steps that are introduced briefly below.

Prediction The selection procedure starts with a simulations of the surrounding vehicles based on currently available sensor information together with the host as it traverses each trajectory. It is again stressed that the interaction with obstacles (if present) are used through a model of the ACC to determine the longitudinal dynamics of the host vehicle. The predicted velocity, acceleration and jerk of the latter as well as a measure of proximity to other road participants are stored at each point along the tentative paths given from the Trajectory Calculation module. A detailed description will be given in 3.4.3.

Feasibility Control To ensure that safety is maintained, the trajectories that are predicted to render the host vehicle too close to an obstacle or outside the road are marked as infeasible and removed from the process. This posterior control is introduced to partly compensate for the removal of the Trajectory Generation module.

Score Attribution Using the predicted point values of the states and proximity measure a score is formed that approximates the objective J in Equation 3.1 for each remaining trajectory. A note here is that the original problem by convention was formed as a minimization, but that the maximization of the inverse of J have been used in the practical implementation. Further details around the construction and calculation of the score will be presented in 3.4.1.

Filtering For practical reasons the raw value of the score are unsuitable to use directly due to which a low pass filter is applied. This implies that previous scores affects the filtered value of a trajectory as it is recomputed and evaluated at the next iteration. A description of the filter used and motivations to its usage will be presented in 3.4.2.

Selection During the selection step, the trajectory with the currently highest filtered score is retained and passed on for execution. All other trajectories are discarded.

3.2 Trajectory Generation for Highway Manoeuvres

As noted in section 2.3 high levels of lateral jerk is the prime contributor to passenger discomfort. Because of this any trajectory generator for use in commercial automotive applications need to assert limits on the jerk levels, and ideally find solutions that minimizes the quantity over the entire trajectory. Versatile and accurate approaches have been proposed by several authors, where an optimal control problem is solved on-line, when the trajectory is needed, using a kinematic vehicle model and a cost functional that could be designed to include lateral jerk [18]. Other authors highlight the possibility to do similar calculations off-line and retrieve the trajectories from lookup tables during on-line operation [5]. Although accurate, both approaches suffer from complexity issues and is hard to realize on the hardware platforms that are expected to be available on commercial vehicles in the near future. The former is troubled with high time complexity due to its use of numerical optimal control solvers, while the memory storage requirements might limit the latter. A well known idea proposed by several researchers, e.g. [16][14][19], is to simplify the same problem by decoupling the longitudinal and lateral motions of the vehicle. The evolution of corresponding trajectories can then be computed separately, turning a hard differentially constrained optimization problem to two simpler ones. This of course means disregarding the non-holonomic constraints. In the formulation presented here, it amounts to approximating the vehicle dynamics with that of a holonomic point mass. It is evident that this indeed is a highly inaccurate model in the general situation, but it has been successfully implemented at higher speeds [17]. This model simplification is further examined in 4.1.3 where conditions under which the approximation is valid are detailed. Regarding their computational weight, the simplified formulation of the problem is very attractive as it, in many cases, have a closed form solution which effectively changes the on-line part of the trajectory generation to a low number of calls to very simple functions.

3.2.1 Derivation of optimal trajectory

To generate these trajectories the lateral position of a particle is described as the triple integration of the jerk, j. Being a common comfort measure found in the literature, the latter is also included in the cost functional according to

$$J = \int_0^{t_f} \frac{1}{2} j^2(t) dt$$

for some final time t_f, with the intention of optimizing ride quality over the trajectory. Other comfort measures and thereby cost functionals, are indeed plausible, but findings in psychophysiological research indicates that the minimization of jerk indeed is the best alternative. A further discussion and motivation of the choice can be found in Appendix B.1.

Considering the (lateral) jerk as the control signal $u(t)$, the relevant, unrestricted optimal control problem then becomes:

$$\begin{aligned} \min_{u(t)} \quad & \int_0^{t_f} \frac{1}{2}u^2(t)dt \\ \text{s.t.} \quad & \dot{x}_1(t) = x_2(t) \\ & \dot{x}_2(t) = x_3(t) \\ & \dot{x}_3(t) = u(t) \\ & x_1(0) = x_{10},\ x_2(0) = x_{20},\ x_3(0) = x_{30} \\ & x_1(t_f) = x_{1f},\ x_2(t_f) = x_{2f},\ x_3(t_f) = x_{3f} \end{aligned} \tag{3.3}$$

Here, x_1, x_2 and x_3 is the position, velocity and acceleration respectively, x_{10},x_{20},x_{30} and x_{1f},x_{2f},x_{3f} their values at the initial (set to 0) and final times t_f. Forming the Hamiltonian as

$$H = \frac{1}{2}u^2(t) + \lambda_1 x_2(t) + \lambda_2 x_3(t) + \lambda_3 u(t),$$

where λ_i are the costate variables evolving according to

$$\dot{\lambda}_1 = -\frac{\partial H}{\partial x_1} = 0$$

$$\dot{\lambda}_2 = -\frac{\partial H}{\partial x_2} = -\lambda_1$$

$$\dot{\lambda}_3 = -\frac{\partial H}{\partial x_3} = -\lambda_2$$

giving solutions on the forms

$$\lambda_1 = C_1$$

$$\lambda_2 = -C_1 t + C_2$$

$$\lambda_3 = \frac{1}{2}C_1 t^2 - C_2 t + C_3.$$

The first order optimality conditions then gives the optimal control as

$$\frac{\partial H}{\partial u} = u + \lambda_3 = 0 \ \rightarrow \ u^* = -\lambda_3$$

The corresponding expressions for the states can then be reached by integration of the system dynamics

$$x_3(t) = -\frac{1}{6}C_1 t^3 + \frac{1}{2}C_2 t^2 - C_3 t + C_4 \tag{3.4}$$

$$x_2(t) = -\frac{1}{24}C_1 t^4 + \frac{1}{6}C_2 t^3 - \frac{1}{2}C_3 t^2 + C_4 t + C_5 \tag{3.5}$$

$$x_1(t) = -\frac{1}{120}C_1t^5 + \frac{1}{24}C_2t^4 - \frac{1}{6}C_3t^3 + \frac{1}{2}C_4t^2 + C_5t + C_6 \tag{3.6}$$

where the integration constants C_1,C_2,C_3,C_4,C_5, C_6 are found through the boundary conditions according to

$$C_6 = x_{10} \tag{3.7}$$
$$C_5 = x_{20} \tag{3.8}$$
$$C_4 = x_{30} \tag{3.9}$$
$$C_3 = -\frac{(9x_{30} - 3x_{3f})t_f^2 + (36x_{20} + 24x_{2f})t_f + 60x_{10} - 60x_{1f}}{t_f^3} \tag{3.10}$$
$$C_2 = \frac{(36x_{30} - 24x_{3f})t_f^2 + (192x_{20} + 168x_{2f})t_f + 360x_{10} - 360x_{1f}}{t_f^4} \tag{3.11}$$
$$C_1 = -\frac{(60x_{30} - 60x_{3f})t_f^2 + (360x_{20} + 360x_{2f})t_f + 720x_{10} - 720x_{1f}}{t_f^5} \tag{3.12}$$

The jerk minimizing solution to the trajectory generation problem is thus a quintic polynomial in t, with coefficients depending solely on the boundary conditions and the final time. Similar results have been found in earlier research, c.f [7] and [8]. An example on how the states would develop for a typical set of boundary conditions associated with a lane change is shown in figure 3.8. Note that the independent variable is time and not longitudinal position, making the results scalable w.r.t. the latter. Thus, the trajectory in a spatial coordinate system (s,l) is

$$s = \int_0^{t_{max}} V_{lon}(t)dt \tag{3.13}$$
$$l = -\frac{1}{120}C_1t^5 + \frac{1}{24}C_2t^4 - \frac{1}{6}C_3t^3 + \frac{1}{2}C_4t^2 + C_5t + C_6 \tag{3.14}$$

Here, the coordinates are named according to common practice in road bound path planning, where the longitudinal coordinate, s is taken along the road and named *station* while the lateral, l is taken to be perpendicular to the latter and named *latitude*.

However, there are other important aspects besides the lateral jerk that needs to be accounted for. Ideally any trajectory generator should be able to guarantee that the states are kept within certain levels, most importantly this regards acceleration and position. Unfortunately, there are some difficulties associated with this that reduces the efficiency of the overall approach. When imposed, the inclusion of constraints in general implies that the optimal control problem lacks analytical solution. As an alternative, a numerical method to guarantee bounded acceleration has therefore been implemented that posteriorly modifies the unrestricted solution. In short, the used algorithm iteratively adjusts the final time t_f until the maximum acceleration in Equation 3.4 are confined to lie within some limited bound, i.e. $|x_3(t)| \leq a_{max}\ \forall t$ and some $a_{max} > 0$. Further motivation and detailed descriptions of this procedure is found in Appendix B.2

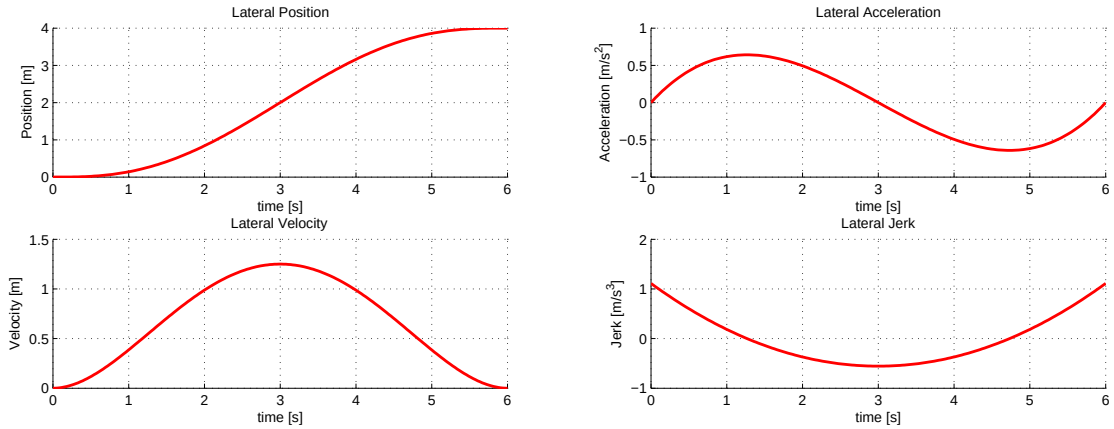


Figure 3.8: State Evolution for a typical lane change manoeuvre. Computed with $x_{10} = 0, x_{20} = 0, x_{30} = 0, x_{1f} = 4, x_{2f} = 0, x_{3f}$ and $t_f = 6$

3.2.2 Definition of Trajectory Endpoints

Lateral Direction The selection system presented in 3.4 functions on a set of trajectories generated from the current position to different endpoints (x_f,y_f). From 3.2 it is seen that y_t can, and need to be, specified explicitly, as it constitutes one of the boundary conditions in the trajectory generation. One of the design choices is thereby where in the lateral coordinates these are placed. Due to common practice in highway driving and safety, the lane centres on the current road are always taken as possible targets. Similarly to what is proposed in [14], additional targets at equally spaced distances from all lane centres are added, allowing the vehicle to perform swerves around partially obscuring obstacles. The latter is useful when larger vehicles are passed and an increased safety margin is desired. The lateral endpoints including both centre and swerve positions is exemplified in Figure 3.9. The remaining boundary conditions needed for the trajectory generator in 3.2, i.e. those on speed and acceleration are set to be stationary at all endpoints. Initial conditions are taken to be the currently retrieved state estimates given from sensor readings.

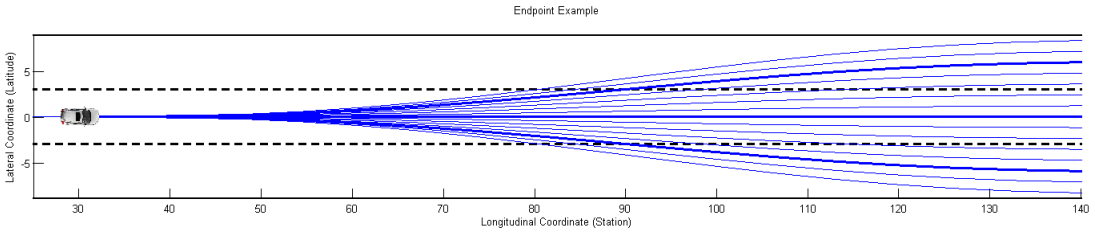


Figure 3.9: Visualization of trajectory with *center* (bold) and *swerve* (thin) endpoints in the lateral direction.

Longitudinal Direction The usage of the built in ACC makes the endpoint placement in the longitudinal direction more complicated, as the trajectory in this direction is given through simulation using the already computed lateral motion. The only possibility for controlling the longitudinal motion is therefore through the boundary conditions of the lateral, i.e through t_f and y_f. In principle, iterative schemes where t_f is adjusted to achieve a certain x_f are plausible for most road scenarios, but in general comes with serious disadvantages. Firstly, an extensive need for simulation arises whereby the computational expense might grow significantly. Secondly, the maximum acceleration during a manoeuvre can no longer be guaranteed to remain within specified bounds, as it will increase with decreasing t_f. Besides, it has been noted for example [20] that targets in the longitudinal direction lacks any real significance during regular highway operations, and is meaningful solely in preparation for exits or turns. For these reasons, explicit definitions of x_f are avoided and the position is retrieved implicitly by demanding that $a_{lat} \leq a_{max}$. In detail the procedure is summarized as:

1. Specify the boundary conditions of the lateral trajectory, i.e. the values of $y(0)$, $y(t_f)$,
   $v_{lat}(0)$, $v_{lat}(t_f)$, $a_{lat}(0)$, $a_{lat}(t_f)$.
2. Use Algorithm 5 from Appendix B to retrieve the final time t_f so that $|a_{lat}(t)| \leq a_{max}$ $\forall t \in (0,t_f)$
3. Simulate the Obstacles and vehicle with ACC using the models presented in 3.4.3 starting from the current state, i.e. (x,y) position, velocity and acceleration, as it traverses the computed lateral trajectory
4. Return the simulated $x(t_f)$ as x_f.

In practice some additional details are included. Firstly, there is limited gain in planning beyond sensor (e.g. radar, camera) range as the simulations rapidly loses validity over the prediction horizon. Due to this, any trajectories where x_f lies beyond this point are truncated. It should be noted that this affects the significance of the comparative measure *Terminal Longitudinal Velocity* as the terminal velocity of all trajectories thereby are restricted. Secondly, as some lateral movements can be very small, and therefore lead to short prediction horizons, a limit is imposed so that

$$t_f = \begin{cases} t_f^A, & \text{if } t_f^A > t_f^{lim} \\ t_f^{lim}, & \text{otherwise} \end{cases}$$

where t_f^A denotes the final time resulting from execution of Algorithm 5 in Appendix B.2. Similar to the previous point, this is motivated by the need to restrict the range of values used in the *Terminal Longitudinal Velocity* comparison.

3.2.3 Adaptation to road geometry

The trajectories generated with the method described above are essentially one dimensional, i.e. they only describe the lateral motion with respect to a fixed coordinate system, decoupling the longitudinal motion. This means that the trajectories needs the presence of a reference line along the direction of the road to make any sense. Due to this there is no possibility for this approach to handle more unstructured scenarios like the zone operation mentioned in 2.3.2. For on-line planning the sensory system of the vehicle consequently must provide the planner with information about the road geometry up to some distance ahead of the current position, including curvature. Following the work done in [17] and [14] the proposed planner executes the paths in a moving coordinate system known as the Frenet frame. Here, the vectors spanning $\mathbb{R}^2$ are formed from the tangent and normal of the curve describing the road geometry, giving the possibility to form the position in some fixed coordinate system as

$$\mathbf{x} = \mathbf{r}(s) + d(s)\mathbf{n}(s)$$

where $\mathbf{x}(s)$ is the position in some fixed (x,y)-coordinate system, s the arclength from some start position, $\mathbf{r}(s)$ the vector following the reference curve, $\mathbf{n}(s)$ the normal vector of said curve and $d(s)$ the planned lateral distance, see Figure 3.10 for further details.

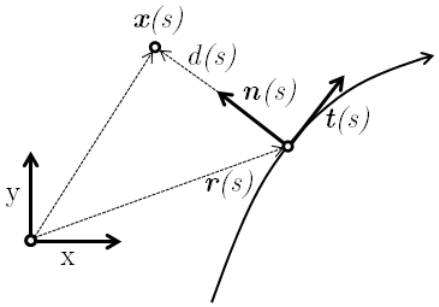


Figure 3.10: Relationship between dynamic and static coordinate systems.

In this way, a path planned in the (s,l) coordinate system is easily "wrapped" around any given reference curve by equating the arclength with the station coordinate, and the normal distance $d(s)$ to the latitude coordinate. A schematic view of this is presented in Figure 3.11

A final issue to note is that the point mass model of the vehicle dynamics fails for obvious reasons to describe the orientation of the vehicle. In accordance with the work done by Werling et. al. [17] this is however neglected with the motivation that given sufficiently high speeds, the deviation between the vehicle and road direction will remain small, even during manoeuvres, if only regular road bound operation is considered. Note that the

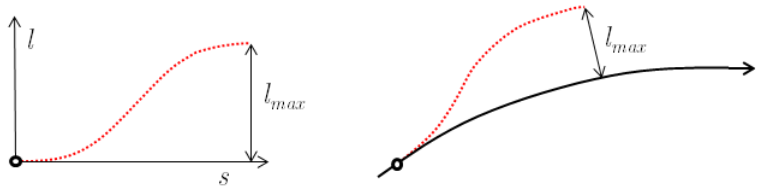


Figure 3.11: Left: Planned trajectory in the (s,l) coordinate system. **Right:** Planned trajectory wrapped around a reference curve.

reference curve, i.e. the representation of the road must be parametrized by arclength for this to work naturally

3.3 Trajectory Generation for Turning Manoeuvres

Contrary to regular road bound operation there is significantly less structure to be used in the planning of turns, making it in nature more similar to the complex *Zone Operation* mentioned in 2.3.2. Luckily however, the planing horizon is usually short, spanning only from the start point of a turn to its corresponding end which means that complex manoeuvres such as those performed while parking can be avoided. Further the longitudinal velocity is normally either held constant or varies slowly, which provides additional relaxation. Where the road bound scenarios essentially admitted separate solutions for longitudinal and lateral motion, the general turning scenario permits no such simplification. The inability to access reliable existing motion references similar to the lane markings or road edges on a normal road instead forces any algorithm tasked with the problem to find a solution including *at least* the two spatial coordinates and direction, making the problem three dimensional (or higher) as opposed to the one dimensional formulation in 3.2, sufficient in the highway case. This fact alone implies a significantly increased problem complexity and natural difficulty to synthesise simple analytic solutions. This section presents the means by which these three dimensional trajectories can be generated efficiently using parametric optimal control and numerical techniques.

3.3.1 Derivation of Optimal Trajectory

The method chosen is based on the work done by McNaughton [21] and that by Kelly and Nagy [3] but includes some simplifications and developments. Modelling the vehicle as a *"Soap Box Car"* the equations of motion can with the notation of Figure 3.12 be formulated as

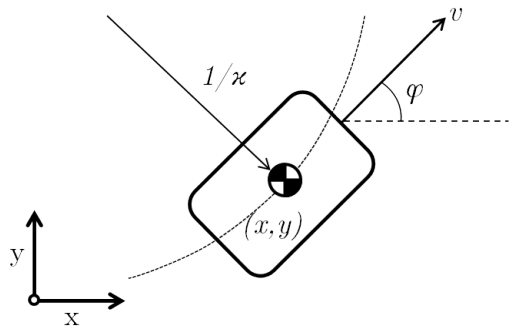


Figure 3.12: Simplified model of vehicle kinematics

$$\begin{aligned} \dot{x}(t) &= v(t)\cos(\varphi(t)) \\ \dot{y}(t) &= v(t)\sin(\varphi(t)) \\ \dot{\varphi}(t) &= v(t)\kappa(t) \end{aligned} \tag{3.15}$$

Noting that the three equations in 3.15 are linear in the longitudinal velocity and defining an arc length parameter s, it is possible to divide them all with $v(t) = ds/dt$ and re-parametrize according to

$$\begin{aligned} \frac{dx}{ds} &= \cos(\varphi(s)) \\ \frac{dy}{ds} &= \sin(\varphi(s)) \\ \frac{d\varphi}{ds} &= \kappa(s) \end{aligned} \tag{3.16}$$

Giving the equations for the states (x,y,φ) at some final length s_f as

$$\begin{aligned} x(s) &= x_0 + \int_0^s \cos(\varphi(\hat{s}))d\hat{s} \\ y(s) &= y_0 + \int_0^s \sin(\varphi(\hat{s}))d\hat{s} \\ \varphi(s) &= \varphi_0 + \int_0^s \kappa(\hat{s})d\hat{s} \end{aligned} \tag{3.17}$$

Due to the couplings between the spatial coordinates and the heading this system can in the general case not be solved in closed form, which necessitates use of numerical techniques. For the purpose of trajectory generation a control function $\kappa(s)$ is now sought after taking the system from an initial state (x_0,y_0,φ_0,κ_0) to a destination state (x_1,y_1,φ_1,κ_1), possibly while optimizing some measure $L(x(s),y(s),\varphi(s),\kappa(s))$ over the in-between trajectory. Formally this can then be written using Equations 3.17 and 3.16

as

$$\begin{aligned}
\min_{\kappa(s),s_f} \quad & \int_0^{s_f} L(x(s),y(s),\varphi(s),\kappa(s))ds \\
\text{s.t.} \quad & \frac{dx}{ds} = \cos(\varphi(s)) \\
& \frac{dy}{ds} = \sin(\varphi(s)) \\
& \frac{d\varphi}{ds} = \kappa(s) \\
& x(0) = x_0, \qquad\qquad x(s_f) = x_1 \\
& y(0) = y_0, \qquad\qquad y(s_f) = y_1 \\
& \varphi(0) = \varphi_0 \qquad\qquad \varphi(s_f) = \varphi_1 \\
& \kappa(0) = \kappa_0 \qquad\qquad \kappa(s_f) = \kappa_1
\end{aligned} \tag{3.18}$$

The general problem, as presented above, is however not directly solvable, since its is searched for directly in the space of all possible functions $\kappa(s)$. A common remedy for problems of this type is to assume some parametrized form of the control function and perform the search in the corresponding parameter space. Kelly and Nagy [3] notes that a certain form, by them named polynomial spirals, is especially suited for the task of solving problems of the type presented in Equation 3.18. They show that their proposition has the representative power to cover most situation arising in a limited horizon path planning context. Specifically, these functions are polynomials in arclength of arbitrary order giving a curvature function for Equation 3.16 on the form

$$\kappa(s) = a + bs + cs^2 + ds^3 + es^4 + \dots$$

Following this approach, the prescribed form limits the search space to that of the polynomial coefficients and the terminal arclength s_f, reducing the infinite dimensional optimization problem to a finite $n+1$-dimensional one, where n is the order of the polynomial. Collecting the polynomial parameters in the vector $\mathbf{p} = [a,b,c,...]$ Equation 3.18 can be restated as

$$\begin{aligned}
\min_{\mathbf{p},s_f} \quad & \int_0^{s_f} L(\mathbf{p},s)ds \\
\text{s.t.} \quad & \frac{d\mathbf{x}}{ds} = \mathbf{f}(\mathbf{p},s) \\
& x(\mathbf{p},0) = x_0, \quad x(\mathbf{p},s_f) = x_1 \\
& y(\mathbf{p},0) = y_0, \quad y(\mathbf{p},s_f) = y_1 \\
& \varphi(\mathbf{p},0) = \varphi_0 \quad \varphi(\mathbf{p},s_f) = \varphi_1 \\
& \kappa(\mathbf{p},0) = \kappa_0 \quad \kappa(\mathbf{p},s_f) = \kappa_1
\end{aligned} \tag{3.19}$$

Here, the system dynamics as stated in Equation 3.16 has been collected in $d\mathbf{x}/ds = \mathbf{f}(\mathbf{p},s)$ with the state layout $\mathbf{x} = [x,y,\varphi]$. Though simplified, the problem presented in

Equation 3.19 is still problematic. However, by placing the coordinate system at the desired startpoint and aligning the abscissa with the initial heading, further reduction is possible and the problem can be written

$$\begin{aligned} \min_{\mathbf{p},s_f} \quad & \int_0^{s_f} L(\mathbf{p})ds \\ \text{s.t.} \quad & \frac{d\mathbf{x}}{ds} = \mathbf{f}(\mathbf{p},s) \\ & x(\mathbf{q}) = x_1 - x_0, \quad y(\mathbf{q}) = y_1 - y_0 \\ & \kappa(\mathbf{p},0) = \kappa_0 \qquad\quad \kappa(\mathbf{p},s_f) = \kappa_1 \\ & \varphi(\mathbf{q}) = \varphi_1 - \varphi_0 \end{aligned} \tag{3.20}$$

Furthermore, as the control function is available in explicit form the integration of the system dynamics, i.e. Equation 3.17, is decoupled whereby they can be substituted into the cost and boundary conditions and the problem rewritten

$$\begin{aligned} \min_{\mathbf{p},s_f} \quad & \int_0^{s_f} \hat{L}(\mathbf{p},s)ds \\ \text{s.t.} \quad & \int_0^{s_f} \cos(\varphi(\mathbf{p},s))ds - x_1 + x_0 = 0 \\ & \int_0^{s_f} \sin(\varphi(\mathbf{p},s))ds - y_1 + y_0 = 0 \\ & \kappa(\mathbf{p},0) - \kappa_0 = 0 \\ & \kappa(\mathbf{p},s_f) - \kappa_1 = 0 \\ & \varphi(\mathbf{p},s_f) - \varphi_1 + \varphi_0 = 0 \end{aligned} \tag{3.21}$$

As an additional consequence of the assumed control function form, the boundary conditions on curvature $\kappa(s)$ and heading $\varphi(s)$ are linear in the polynomial coefficients, i.e.

$$\begin{aligned} \kappa(\mathbf{p},0) &= a &&= \kappa_0 \\ \kappa(\mathbf{p},s_f) &= a + bs_f + cs_f^2 + ds_f^3 + \ldots &&= \kappa_1 \\ \varphi(\mathbf{p},s_f) &= as_f + \frac{1}{2}bs_f^2 + \frac{1}{3}cs_f^3 + \frac{1}{4}ds_f^4 + \ldots &&= \varphi_1 - \varphi_0 \end{aligned} \tag{3.22}$$

By solving these equations for the coefficients a,b,c and substituting the result back into Equation 3.21 allows a removal of the constraints on heading and curvature from the problem. Further, the problem dimensionality is reduced as the coefficients a,b and c now is completely dependent on the remaining parameters s_f,d,e,f,... and the boundary

conditions κ_0, κ_1 and φ_1. Letting $\hat{\mathbf{p}} = [d,\ e,\ \ldots]$ and with

$$\begin{aligned} g_1(\hat{\mathbf{p}},s_f) = &\quad \int_0^{s_f} \cos(\varphi(\hat{\mathbf{p}},s_f,s))ds - x_1 + x_0 \\ g_2(\hat{\mathbf{p}},s_f) = &\quad \int_0^{s_f} \sin(\varphi(\hat{\mathbf{p}},s_f,s))ds - y_1 + y_0 \end{aligned} \tag{3.23}$$

the final, reduced, problem can then be written

$$\begin{aligned} \min_{\hat{\mathbf{p}},s_f} \quad & J(\hat{\mathbf{p}},s_f) = \int_0^{s_f} \hat{L}(\hat{\mathbf{p}},s_f,s)ds \\ \text{s.t.} \quad & g_1(\hat{\mathbf{p}},s_f) = 0 \\ & g_2(\hat{\mathbf{p}},s_f) = 0 \end{aligned} \tag{3.24}$$

3.3.2 Numerical Considerations

It is stressed that the integrals in $g_1(\hat{\mathbf{p}},s_f)$ and $g_2(\hat{\mathbf{p}},s_f)$ cannot be solved analytically since they are of the *generalized Fresnel* type. Consequently, numerical integration is required. The method of choice for this particular problem is the *Composite Simpson Quadrature rule* due to its lightweight and performance on relatively smooth integrands. No detailed description on how this method operates is given here, but the interested reader finds a short description in Appendix D.4. As a general consequence, solutions to the optimization problem can only be found through a numerical optimization algorithm.

A final note is that the problem presented in Equation 3.24 in general is numerically ill behaved, as its decision variables (independent polynomial coefficients, d,e,$\ldots$ and terminal arclength s_f) typically differs several orders of magnitude in size. Consider as an example the variables d and s_f, which in the normal cases is in the order of 10^{-6} and 10^2 respectively. Given that higher order terms, i.e. e,$f\ldots$, will be even smaller it is quite clear that numerical issues potentially can arise. Following [3] and [21] a variable scaling is therefore performed to bring all decision variables closer in order and thereby improve the convergence and stability properties of the problem. Having essentially no impact on how the solution is obtained, the details of the scaling procedure is omitted here but can be found in Appendix D.3.

3.3.3 Finding a feasible solution

For the task of finding a solution simply satisfying the constraint equations, the objective in Equation 3.24 is ignored and the problem is reduced to a system of two non linear

equations, i.e.

$$\begin{aligned} g_1(\hat{\mathbf{p}},s_f) &= 0 \\ g_2(\hat{\mathbf{p}},s_f) &= 0 \end{aligned} \tag{3.25}$$

where $g_1(\hat{\mathbf{p}},s_f)$ and $g_2(\hat{\mathbf{p}},s_f)$ are defined according to Equation 3.23. Noting that this requires two independent parameters to be uniquely solvable and emphasizing that s_f is included as a decision variable, the necessary control function takes to the form of a fourth order polynomial, i.e.

$$\kappa(s) = a + bs + cs^2 + ds^3 \tag{3.26}$$

where then a,b and c are dependent on d and s_f through the linear system in Equation 3.22. Consequently, $\hat{\mathbf{p}} = d$. By making the substitutions, the assumed control function can be written as

$$\begin{aligned} \kappa(d,s_f,s) = \kappa_0 + &\left(\frac{12(\varphi_1 - \varphi_0) - 4(2\kappa_0 + \kappa_1)s_f + ds_f^4}{2s_f^2} \right) s \\ &- 3\left(\frac{4(\varphi_1 - \varphi_0) - 2(\kappa_0 + \kappa_1)s_f + ds_f^4}{2s_f^3} \right) s^2 + ds^3 \end{aligned} \tag{3.27}$$

It should be noted here that this parametrisation differs from what is presented in [21] and [3], where the same problem (unnecessarily) is solved for three equations and three variables. The results is essentially the same however, with the attractive difference that the formulation presented here admits both easy visualisation and reduced computation.

Defining the squared residual of the two expressions in Equation 3.25 as

$$r(d,s_f) = \frac{1}{2}\left(g_1(d,s_f)^2 + g_2(d,s_f)^2\right) \tag{3.28}$$

the non-linear system can be reformulated to the equivalent minimization problem

$$\min_{(d,s_f)} \quad r(d,s_f) \tag{3.29}$$

Here, all (d,s_f) such that $r(d,s_f) = 0$ also constitutes solutions to Equation 3.24. To solve this problem in an online setting, the Levenberg-Marquardt (LM) Algorithm, as presented in [22], is implemented, motivated mainly by its frequent use in multidimensional root finding. Further details regarding the LM Algorithm will not be given here, but a description including a step-by-step presentation of the implementation is given in Appendix D.1.

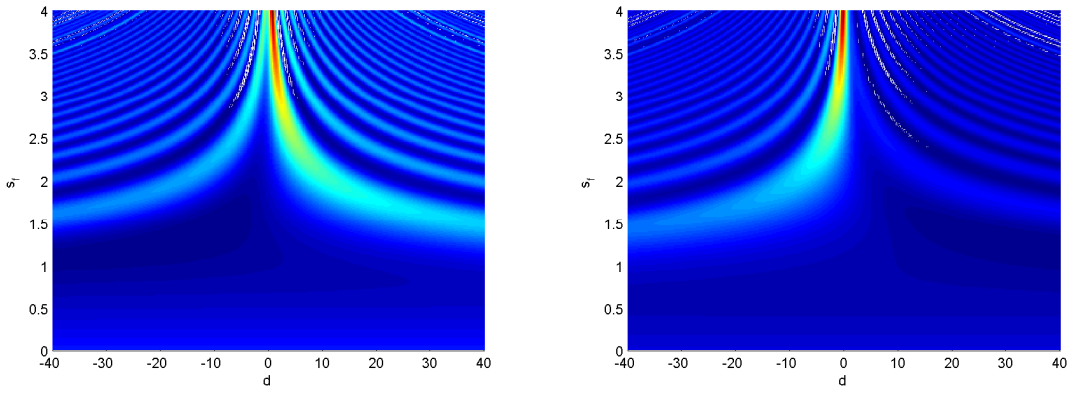
In principle, solutions to Equation 3.29 exists that allows reversed motion, corresponding to $s_f < 0$. Assuming that all turns are made in the forward direction these are highly

undesirable due to which the problem is augmented by adding the requirement $s_f \geq \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$. That is, it is required that the total length is greater than the euclidean distance to the target. Formally, the constraint inclusion is included according to

$$\begin{aligned} \min_{(d,s_f)} \quad & r(d,s_f) \\ \text{s.t.} \quad & s_f \geq \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \end{aligned} \tag{3.30}$$

To handle the introduced inequality, the standard LM-algorithm is augmented with a *projection* component, where the current iterate is projected back on the feasible set if it is placed outside by the step in the previous iteration. Descriptions around the implementation is presented in Appendix D.2.

It is stressed that the optimization problem in Equation 3.29 is dependent on the boundary conditions x_0,x_1,y_0,y_1,θ_0,θ_1,κ_0 and κ_1 through Equations 3.23 and 3.27. Consequently, different boundary conditions gives different solutions in d and s_f as the optimization landscape is changed. A visualization of the dependence is given in Figure 3.13 where the squared residual is drawn for two different sets of boundary conditions. Note especially the placement of the large basin as it shifts from left to right.



(a) (b)

Figure 3.13: Exemplification of optimization landscapes of Equation 3.29. Dark blue and red colors shows low and high values of $r(d,s_f)$ respectively, with in between hues representing intermediate numbers. Note that the numerical values of d and s_f are scaled according to the method described in Appendix D.3. (a) Shows the landscape for $x_0 = y_0 = \theta_0 = \kappa_0 = \kappa_1 = 0, \theta_1 = 120°, x_1 = 15$ and $y_1 = 10$, while (b) shows $x_0 = y_0 = \theta_0 = \kappa_1 = 0, \theta_1 = 0°, \kappa_0 = 0.2, x_1 = 5$ and $y_1 = 5$

3.3.3.1 Generating the Initial Guess

The Levenberg-Marquardt algorithm is like most other efficient optimization algorithms local in nature. In problems like Equation 3.29 it is therefore not only important to initialize the solver close to a minima, but to the *right* minima, to ensure desired results. For sake of clarity it is emphasized that all minima where $r(d,s_f) = 0$ corresponds to parameter combinations for which the spatial boundary conditions are fulfilled, i.e. where the vehicle reaches its desired pose, and represents different ways of achieving this. Figure 3.14 gives examples of paths resulting from two different minima. For both paths shown $r(d,s_f) = 0$ and the boundary conditions are satisfied exactly.

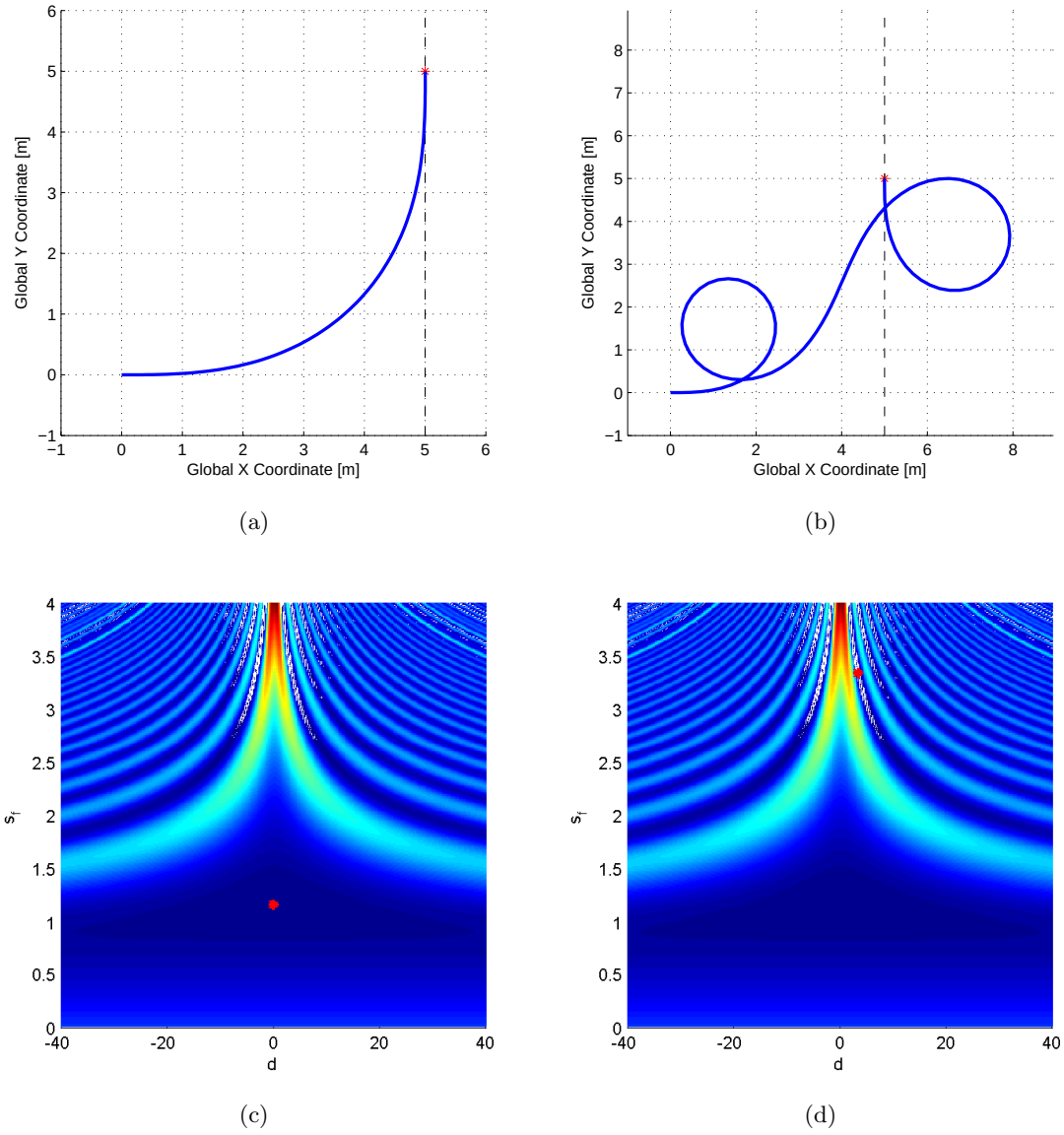


Figure 3.14: Illustration of paths resulting from different Minima of Equation 3.29 using the boundary conditions $\kappa_1 = \kappa_0 = x_0 = y_0 = \varphi_0 = 0, \varphi_1 = 90°, x_1 = y_1 = 5$.(a) A shows the path from the correct minima. The position of this minima in the $r(d,s_f)$ landscape is marked with a red dot in (b). Similarly, the trajectory resulting from another minima is given in (b) and its corresponding position in the $r(d,s_f)$ landscape is shown in (d). Note that both satisfies the boundary conditions.

Kelly and Nagy [3], even though utilizing a slightly different formulation than that presented above, proposes a simple heuristic for generating the starting point of the

numerical optimization. In their work, the equivalents of the free parameter d are set to 0 and the scaled final arc length is defined as a function of the boundary condition φ_1 and initialized to

$$s_{fs} = \varphi_1^2/5 + 1 \tag{3.31}$$

The authors derives this from solutions over a wide envelope, i.e. range, of boundary conditions with some exhaustive search method, followed by *"eyeballing"*. From this simple relationship they claim extremely good performance results in terms of computation, with execution times well under one millisecond on a, by today's standard, slow desktop computer. Even though the simplicity indeed is appealing it has been found that the quality of initial guesses produced this way affects the performance severely, and that their results hardly is reproducible. This applies even when more robust algorithms, like those included in MATLAB's optimization package are used together with precise numerical integration. For a wide scope of initial conditions the algorithm fails to converge, for others it either converges to the wrong minima or requires an unacceptable amount of iterations. As an alternative, the usage of lookup tables is mentioned in [3], where the sought after minima could be found and stored off-line for a number of boundary conditions and the initial guess for in-between values retrieved online through some sort of interpolation. Although not tested, the risk of exploding storage requirements might present a problem for implementation on industrial real time computers, in particular if more than the two parameters (d,s_f) are used and a high resolution is required.

To remedy the accuracy problems and to prevent increased resource consumption an alternative approach is taken, intended to capture the lightweight of the simple heuristic and the performance of a finely grained lookup table. The main idea is to find a function from the boundary condition to the vicinity of the correct minima, by which the iterations needed by the optimization algorithm decreases and its robustness increases. Rewriting the system in Equation 3.25 as functions of both the parameters $\mathbf{q} = [d\ s_f]$ and boundary conditions $\mathbf{b} = [\ \kappa_0,\ \kappa_1,\ x_0,\ x_1,\ y_0,\ y_1,\ \varphi_0,\ \varphi_1]$, all minima, i.e. paths between initial and terminal pose, satisfies

$$\mathbf{g}(\mathbf{b},\mathbf{q}) = [g_1(\mathbf{b},\mathbf{q}),\ g_2(\mathbf{b},\mathbf{q})] = \mathbf{0} \tag{3.32}$$

Letting $F(\mathbf{b}) = \mathbf{q}$ be the implicit, multivalued function[1] defined by the constraint satisfaction, i.e. $F(\mathbf{b}) = \mathbf{q} \mid \mathbf{g}(\mathbf{b},\mathbf{q}) = \mathbf{0}$, that maps a set of boundary conditions $\mathbf{b}$ to *all* minimizing $\mathbf{q}$ of that particular configuration. By this each branch $F^x(\mathbf{b})$ of $F(\mathbf{b})$ is is such that $F^x(\mathbf{b}) : \mathbb{R}^8 \mapsto \mathbb{R}^2$. Letting $F^*(\mathbf{b})$ be the principal branch of $F(\mathbf{b})$, and defining it as the map from the boundary conditions to the *correct* minimum, which is taken as the one with smallest s_f out of all solutions where $s_f \geq \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$. A schematic illustration of the definitions of $F^*(\mathbf{b})$ and $F(\mathbf{b})$ is given in Figure 3.15

[1]The term multivalued function is somewhat of a abuse of notation since it by definition does not satisfy the uniqueness criteria of a proper function. By convention, the term is used to denote functions that maps one input to several outputs, c.f. the square root of x

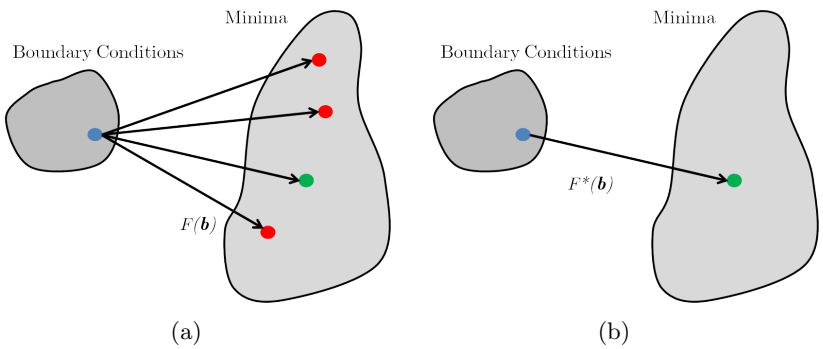


Figure 3.15: Illustration of the definitions of $F(\mathbf{b})$ (a) and $F^*(\mathbf{b})$ (b). The gray areas represent the sets of boundary conditions and corresponding minima, the red dots the undesired minima and the green dot the desired minimum

For obvious reasons $F^*(\mathbf{b})$ cannot be specified analytically, but would if available be the solution to Equation 3.25. Knowing this, an approximation of $F^*(\mathbf{b})$ within some tolerance would form a perfect replacement for the presented simple heuristic.

The *Universal Approximation Theorem (UAT)* states that there exists constants n, b_i, w_{ij}, α_i where $i = 1...n$ and $j = 1...m$, such that all functions $G \in C^1$ on the hypercube $[0,1]^m$ can be approximated by a structure on the following form

$$\hat{G}(x_1,...,x_m) = \sum_{i=1}^{n} \alpha_i g \left(b_i + \sum_{j=1}^{m} w_{ij} x_j \right) \tag{3.33}$$

so that

$$|G(x_1,...,x_m) - \hat{G}(x_1,...,x_m)| < \varepsilon \tag{3.34}$$

for all $\varepsilon > 0$, given that the function $g(x)$ is bounded, non-constant and monotonically increasing. For sake of clarity it should be noted that it is trivial to find an affine mapping from any *bounded* part of the domain of G to $[0,1]^m$, and that use of such transformations implies that the theorem then holds only for some subset of that domain. Further details or proofs regarding this results are not given here but can be found in for example [23].

Single layer Feed Forward Neural Networks (FFNN) with sigmoidal activation functions is a framework which has been shown to realize this, and is therefore well suited to approximate F^*, given an adequate number of neurons and trained with, i.e. has had the parameters b_i, w_{ij} and α_i adjusted, a large enough and a sufficiently rich data set. The process is essentially that of fitting a non linear function to the given data. As with other techniques, extrapolations too far outside the original data set suffers the risk of being inaccurate and the FFNN will therefore, at best, only be able to provide adequate approximations over some subset of F^*'s domain. Fortunately, this is enough

since the interesting cases, i.e. different versions of normal turning, restricts the envelope of boundary conditions considered greatly.

The training data is generated by solving $\mathbf{g}(\mathbf{b},\mathbf{q}) = \mathbf{0}$ for the *correct* minima over this envelope and used to adapt the approximation $\hat{F}^*$ if convergence to the correct minima occurs. The boundary conditions $\mathbf{b}$ used are considered inputs in the adaptation process and the outputs in (d,s_f) targets, schematically illustrated in Figure 3.16.

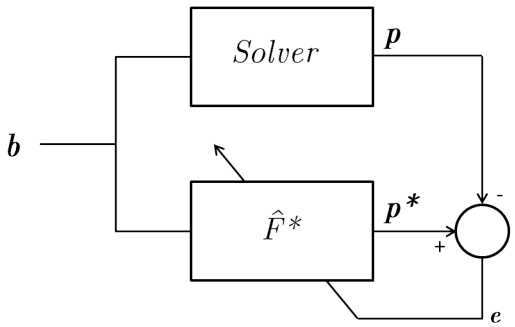


Figure 3.16: Schematic illustration of the adaptation (training) of the approximate function $\hat{F}^*$. The *Solver* block contains a robust, off-line optimization procedure.

The *UAT* is an existence theorem and does not state anything on how the parameters of the approximation are to be retrieved. Based on adaptation using a finite data set, the FFNN will therefore be trained on a collection of nearby elements in the subset where the approximation is sought after. Although evaluations of the retrieved approximation outside the dataset used for adaptation can be coupled with poor performance, the benefit of using FFNN, or any other fitting technique for that matter, is its ability to *generalize* somewhat on the data given. This means that the approximation will, within some tolerance level, hold in a vicinity of the training data, allowing the resulting function to perform adequately for input data different than that with which it was trained. In essence, this makes it possible to capture the behaviour of the unknown function over a certain range by adapting the approximation with a finite collection of input-output pairs, as visualized schematically in Figure 3.17

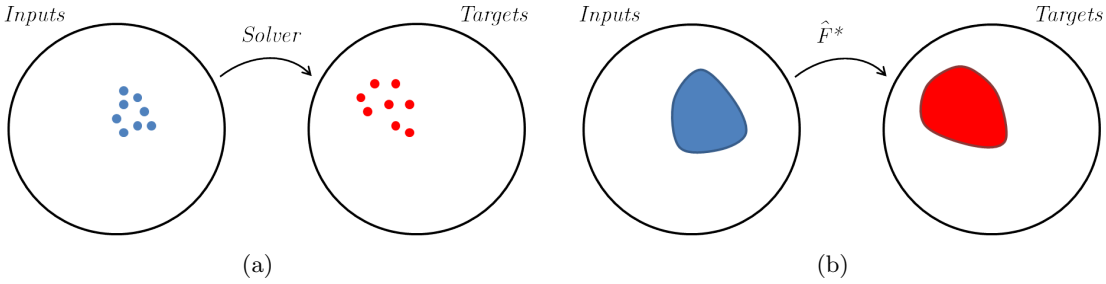
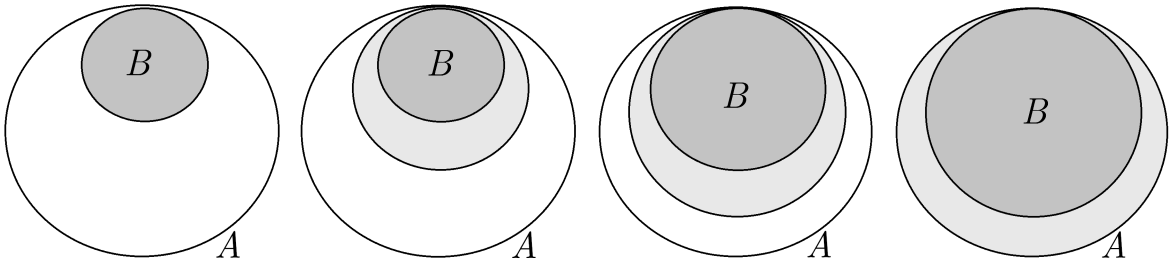


Figure 3.17: Schematic illustration of the generalization process. (a) The unknown function is evaluated at a number of points which with the retrieved results form the basis for training $\hat{F}^*$.(b) The resulting approximation holds within some accuracy for a wider set of inputs.

Based on this, the main idea is the following: An initial data set for training of the network is generated using the successful results of some robust optimization method with highly accurate numerical integration for a given envelope of initial conditions. For this first data set the simple heuristic proposed in [3] is used to provide the initial guess, and convergence will in general occur if this places it in the basin of attraction of the correct minima. When a certain number of points successfully have been added to the dataset, it is used to train a FFNN. With the ability to generalize, the resulting approximation of F^* now gives outputs that ends up in the attractive region to a larger extent of than the simple heuristic. After this, the process is iterated repeatedly with the previously trained Neural Network as provider of initial guesses. Given that the network is able to generalize somewhat in each iteration, the subset of the domain where the network approximates F^* good enough will increase with each step thus leading to a higher success rate of the optimization procedure in the next iteration, as illustrated in Figure 3.18



(a) (b) (c) (d)

Figure 3.18: Illustration of the performance of the iterated fitting proposed. The set boundary condition envelope, A is shown as a black bordered white circle. Its subset that sing the currently avaliable starting point provider leads to correct convergence is shown in dark grey, whereas the improvement that is visible in the next iteration is represented by the light gray regions. In (a) the result of a poorly chosen starting point is visualized that is augmented after the first iteration (b). As the procedure is repeated in (c) and (d) the performance is improved due to better and better initial guesses

All of this is of course time consuming and therefore done off-line. The result however, is a lightweight function that within specified bounds, i.e. those of the envelope used in training, returns a highly accurate initial guess giving the lightweight optimization procedure described in 3.3.3 the possibility to safely reach its target in a very low number

of iterations. The procedure of generating this function is detailed in Algorithm 1.

Data: Envelope of boundary conditions E, number of boundary condition combinations to use $n_{\mathbf{B}}$
Result: Function $f_I(\mathbf{B}$ approximating the mapping between the boundary condition $\mathbf{B}$ and the correct minima $\mathbf{q}_m$
$f_I(\mathbf{B}) \leftarrow$ @($\mathbf{B}$) SimpleHeuristic($\mathbf{B}$) ;
while *Success rate* $r < r_{lim}$ **do**
  $S_{\mathbf{B}} \leftarrow$ GenerateSetOfBoundaryConditions(E,$n_{\mathbf{B}}$) ;
  $Inputs \leftarrow \emptyset$;
  $Targets \leftarrow \emptyset$;
  $n_{succeeded} \leftarrow 0$;
  foreach $\mathbf{B} \in S_B$ **do**
    $\mathbf{q}_0 \leftarrow f_I(\mathbf{B})$;
    $\mathbf{q}_m \leftarrow$ RobustOptimization($\mathbf{q}_0, \mathbf{B}$);
    if *optimization succeeded and correct minima is found* **then**
      $Inputs \leftarrow$ AddBoundaryConditionToInputs($\mathbf{B}$,$Inputs$);
      $Targets \leftarrow$ AddMinimaToTargets($\mathbf{q}_m, Targets$);
      $n_{succeeded} \leftarrow n_{succeeded} + 1$
    end
  end
  $Net(\mathbf{B}) \leftarrow$ @($\mathbf{B}$) TrainNetwork($Inputs$,$Targets$);
  $f_I(\mathbf{B}) \leftarrow Net(\mathbf{B})$;
  $r \leftarrow$ GetSuccessRate($n_{succeeded}$,$n_{\mathbf{B}}$);
end
return $f_I(\mathbf{B})$

Algorithm 1: Algorithm for generating the Neural Network function that provides initial guesses to the optimization problem in Equation 3.30. Note that the functions specified with a leading @ are function handles, and the assignments are to be read as those valid for such. The method GenerateSetOfBoundaryConditions(E,$n_{\mathbf{B}}$) generates a set of $n_{\mathbf{B}}$ selected from the envelope E while AddBoundaryConditionToInputs($\mathbf{B}$,$Inputs$) and AddMinimaToTargets($\mathbf{q}_m, Targets$) simply adds a element to the collections $Inputs$ and $Targets$ respectively. The networks are built and trained in TrainNetwork($Inputs$,$Targets$) using MATLAB's Neural Network toolbox which includes pre- and post processing of the inputs and outputs respectively.

The structure used is a singled layered perceptron network with hypertangent and linear activation functions in the hidden and output layers respectively. The necessary parameters, weights biases etc., are easily retrieved from the resulting MATLAB neural network object for standalone implementation when the algorithm has completed. Given that space here is limited a detailed exact descriptions of Neural Networks will not be given, the reader is instead referred to any text book covering the topic, for instance [23].

3.3.4 Introducing Optimality

Since the solution method presented in the previous section only admits solutions to the basic problem of finding *a* path between two points in a three dimensional configuration space it cannot give any statements on its optimality, regardless on how it is metered. With the proposed parametrization the solutions to the system in Equation 3.25 are a collection of intersection points between two dimensional curves, whereby any internal ranking by necessity is discrete. Drawing from the experiences gained in 3.3.3.1 it can be assumed that most of these correspond to unnatural and undesired motions, essentially leaving one feasible option. To render the merely resulting trajectories optimal in some sense, more freedom in the assumed control function is needed. Where the feasible case contained two independent variables for determination of the polynomial coefficients in Equation 3.26, the introduction of additional coefficients neatly accomplishes this feat. Using the three independent variable case to exemplify, the control function is then defined as

$$\kappa(s) = a + bs + cs^2 + ds^3 + es^4 \tag{3.35}$$

As in Equation 3.27 the boundary conditions linear in the coefficients are included using Equation 3.22 which gives that the independent parameters are (d,e,s_f). The problem now obviously lacks a unique solution, as it is comprised of two equations with three independents. The two relations in Equation 3.25, which in the two variable case could be interpreted as curves on the plane and their intersections as points, can now be viewed as surfaces in (d,e,s_f) and their intersection as three dimensional curve segments. Due to this, an infinite number of combinations along the intersection curves now satisfies the spatial boundary conditions, and a specific selection can be based on a given optimality criteria. In higher dimensions, i.e. using more free parameters, analogue reasoning extends the concept to hyper surfaces.

3.3.4.1 Solution including a cost Functional

Considering the problem stated in Equation 3.24 with $\mathbf{g}(\hat{\mathbf{p}},s_f) = [g_1(\hat{\mathbf{p}},s_f),\ g_2(\hat{\mathbf{p}},s_f)]$, and defining its Hamiltonian as

$$H(\hat{\mathbf{p}},s_f) = J(\hat{\mathbf{p}},s_f) + \mu^\top \mathbf{g}(\hat{\mathbf{p}},s_f) \tag{3.36}$$

By letting $\mathbf{q} = [\hat{\mathbf{p}},\ s_f]$The first order optimality conditions, i.e. Karush-Kuhn-Tucker (KKT) conditions, then states that

$$\frac{\partial H}{\partial \mathbf{q}} = \frac{\partial J(\mathbf{q})}{\partial \mathbf{q}} + \mu^\top \frac{\partial \mathbf{g}}{\partial \mathbf{q}} = 0 \tag{3.37}$$

$$\frac{\partial H}{\partial \mu} = \mathbf{g}(\mathbf{q}) = 0 \tag{3.38}$$

must be fulfilled at any extreme point of the constrained problem [22]. The μ here denotes the Lagrange multipliers, and $\mu \geq 0$ has to hold at an optimal point. As demonstrated by

Kelly and Nagy [3] the decision variables $\mathbf{q}$ and multipliers μ can be solved for directly through 3.37 and 3.38 using using the same multidimensional root finding techniques introduced in 3.3.3. As noted in Appendix D.1, the underlying Levenberg Marquardt Algorithm is based on repeated linearisation around the current iterate, in this case momentary values of $\mathbf{q}$ and μ. The linearization of the KKT conditions in Equations 3.37 and 3.38 around ($\mathbf{q}$,μ) is expressed as

$$\begin{aligned}\left.\frac{\partial H}{\partial \mathbf{q}}\right|_{\mathbf{q}+\boldsymbol{\Delta}\mathbf{q},\mu+\boldsymbol{\Delta}\mu} \approx & \left.\frac{\partial H}{\partial \mathbf{q}}\right|_{\mathbf{q},\mu} + \left.\frac{\partial^2 H}{\partial \mathbf{q}^2}\right|_{\mathbf{q},\mu} \boldsymbol{\Delta}\mathbf{q} + \boldsymbol{\Delta}\mu^\top \left.\frac{\partial^2 H}{\partial \mu \partial \mathbf{q}}\right|_{\mathbf{q},\mu} \\ = & \left.\frac{\partial H}{\partial \mathbf{q}}\right|_{\mathbf{q},\mu} + \left.\frac{\partial^2 H}{\partial \mathbf{q}^2}\right|_{\mathbf{q},\mu} \boldsymbol{\Delta}\mathbf{q} + \boldsymbol{\Delta}\mu^\top \left.\frac{\partial \mathbf{g}}{\partial \mathbf{q}}\right|_{\mathbf{q}} = 0\end{aligned} \tag{3.39}$$

$$\begin{aligned}\left.\frac{\partial H}{\partial \mu}\right|_{\mathbf{q}+\boldsymbol{\Delta}\mathbf{q},\mu+\boldsymbol{\Delta}\mu} \approx & \left.\frac{\partial H}{\partial \mu}\right|_{\mathbf{q},\mu} + \left.\frac{\partial^2 H}{\partial \mu \partial \mathbf{q}}\right|_{\mathbf{q},\mu} \Delta\mathbf{q} + \boldsymbol{\Delta}\mu^\top \left.\frac{\partial^2 H}{\partial \mu^2}\right|_{\mathbf{q},\mu} \\ = & \mathbf{g}(\mathbf{q}) + \left.\frac{\partial \mathbf{g}}{\partial \mathbf{q}}\right|_{\mathbf{q}} \boldsymbol{\Delta}\mathbf{q} = 0\end{aligned} \tag{3.40}$$

with the Hamiltonian Hessian according to

$$\left.\frac{\partial^2 H}{\partial \mathbf{q}^2}\right|_{\mathbf{q},\mu} = \left.\frac{\partial^2 J}{\partial \mathbf{q}^2}\right|_{\mathbf{q}} + \mu_1 \left.\frac{\partial^2 g_1}{\partial \mathbf{q}^2}\right|_{\mathbf{q}} + \mu_2 \left.\frac{\partial^2 g_2}{\partial \mathbf{q}^2}\right|_{\mathbf{q}} \tag{3.41}$$

Transposing Equation 3.39 allows a compound matrix equation to be defined, capturing both relationships

$$\begin{bmatrix} \left.\frac{\partial^2 H}{\partial \mathbf{q}^2}\right|_{\mathbf{q},\mu} & \left(\left.\frac{\partial \mathbf{g}}{\partial \mathbf{q}}\right|_{\mathbf{q}}\right)^\top \\ \left.\frac{\partial \mathbf{g}}{\partial \mathbf{q}}\right|_{\mathbf{q}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta}\mathbf{q} \\ \boldsymbol{\Delta}\mu \end{bmatrix} = \begin{bmatrix} -\left.\frac{\partial H}{\partial \mathbf{q}}\right|_{\mathbf{q},\mu} \\ -\mathbf{g}(\mathbf{q}) \end{bmatrix} \tag{3.42}$$

Denoting the left hand side matrix A and its right hand counterpart B, the Levenberg Marquardt Algorithm is applied based on

$$\begin{bmatrix} \boldsymbol{\Delta}\mathbf{q} \\ \boldsymbol{\Delta}\mu \end{bmatrix} = \left(A^\top A + \lambda I\right)^{-1} B \tag{3.43}$$

$\lambda > 0$ is an algorithmic parameter that is further explained in Appendix D.1. It should be noted here that a procedure like the one presented here is analogous to performing a sequential quadratic program on the equality constrained minimization of the squared residual $r(\mathbf{q})$.

3.3.4.2 Initial Guess

In essence Algorithm 6 of Appendix D.1 is applied as it is, with the main difference to the merely feasible case being the method with which the initial guesses are generated. Although a similar neural network based approach is possible, it is by necessity bound to specific cost functions $J(\mathbf{q})$ and thereby somewhat inflexible, due to which it has been avoided. In further accordance with [3], the initial values of the introduced free parameters are instead set to zero and the (d,s_f) are initiated as the solution to the basic problem as calculated for the same boundary conditions. Using these values, the multipliers μ at the zero'th iterate are found approximately by a least squares solution of Equation 3.37, that is, as

$$\mu = -\left(\frac{\partial \mathbf{g}}{\partial \mathbf{q}}\right)^{\dagger} \frac{\partial J(\mathbf{q})}{\partial \mathbf{q}} \tag{3.44}$$

where † denotes the left pseudo inverse, i.e. $M^{\dagger} = (M^{\top}M)^{-1}M^{\top}$. Geometrically the procedure then starts from a feasible point in the (d,s_f,0,0,...) cross section of (d,s_f,e,f,...) space from which it moves out in vicinity of the hypersurface formed by the intersection of $g_1(\mathbf{q}) = 0$ and $g_2(\mathbf{q}) = 0$. As such, its necessarily prone to converge to the closest minima, meaning that others indeed might exist. It is however unlikely that the global minima is found from starting points other than the one found through the basic solution, since the general shape is preserved between the basic and optimized case. Re-examination of Figure 3.14(b) illustrates the "knots" that appear at all other solutions to the basic problem, which regardless of the choice of $J(\mathbf{q})$ is highly undesirable. By this, convergence to nearby minima will at least guarantee that the retrieved trajectory is better than that given from the basic case, although it cannot state any guarantees on global optimality.

3.3.4.3 Lateral Acceleration in the Cost Functional

Even though a multitude of of cost measures are plausible, only one is examined due to the limited space in this thesis. Following the simple model presented in Equation 3.16, the lateral acceleration is formed as

$$a_{lat} = v_{lon}^2 \kappa$$

where v_{lon} is the longitudinal velocity. Holding this constant over the trajectory, a measure can then be formed as

$$J = \int_0^{s_f} a_{lat}^2(s)ds = v_{lon}^4 \int_0^{s_f} \kappa^2(s)ds \tag{3.45}$$

by which the longitudinal velocity can be dropped completely. Trajectories minimizing this measure should as a consequence be performed somewhat more nicely, with less curvature.

It should be noted that this choice is somewhat arbitrary and is only meant to demonstrate the usage of cost functionals. In particular, costs that does not demand use of numerical integration are highly desirable. Firstly, overall complexity is reduced, secondly J and its relevant derivatives then has an analytical form. In the parametrization used here, this equates to solely using the heading and its derivatives either standalone or combined, without involvement of the position states.

3.4 Trajectory Selection

In this section, the subsystem responsible for selection between the generated trajectories is detailed. Firstly, the environment is predicted, described in 3.4.3, secondly each tentative trajectory is evaluated using the prediction and associated with a score, and lastly the best trajectory is selected and passed on for execution.

The problem in Equation 3.1 which the planner is intended to approximate, is originally posed as a minimization. However, the selection is for implementation reasons reformulated as the *maximisation* of an objective, i.e. score. Given the adequate invertation of J, the reformulated problem is of course equivalent to a minimization approximation of Equation 3.1. The notion of *cost* is somewhat misleading in the maximization context, is therefore replaced with the more suiting term *utility*. Consequently, Each trajectory is at each time instant associated with a utility value, where the highest scoring is selected and executed. Figure 3.19 exemplifies the development of these values over several algorithm iterations for a generic overtake scenario.

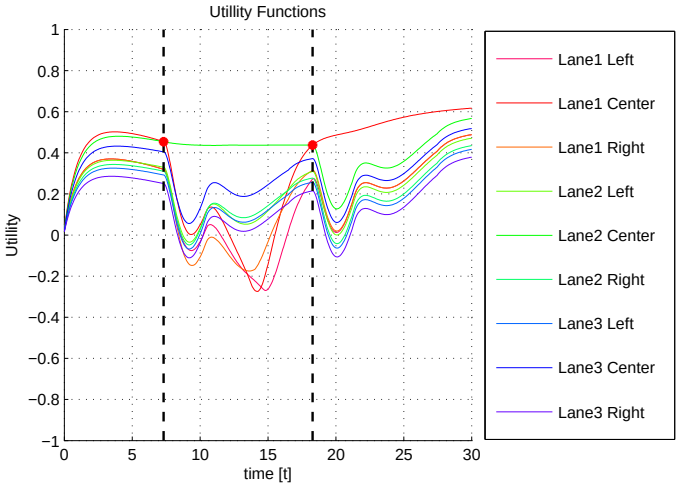


Figure 3.19: Development of the utility functions of each tentative trajectory as a function of time during a simulated overtake scenario. The right hand side legend shows the lateral position of the trajectory endpoints, which is kept constant over time. Trajectory switches occurs when the utility of a particular trajectory has grown enough to exceed that of the trajectory that is currently active. For clarity, these instances are marked with dashed lines and red dots.

3.4.1 Score Attribution

The score value attributed to each trajectory is as stated intended to approximate the objective J in Equation 3.1. For this purpose, it needs to be constructed so that it both captures the desired behaviour of the vehicle (c.f. the requirements stated in 1.1) and enables identification of the best trajectory in the generated set. Following ideas presented in [16] and [14] the approximate utility $\hat{J}$ is taken to be a weighted sum of parts. Letting l be the number of trajectories to be evaluated, $T_m(t)$ a representation of trajectory m, where $1,2,\ldots,m,\ldots,l$ calculated with the information available at time instance t. Further, using $E(t)$ a representation of the predicted environment and with n the number of partial utility values, the summation is done according to

$$\hat{J}_m(t) = \sum_{i=0}^{n} f_i(T_m(t),E(t))\gamma_i \tag{3.46}$$

Here, the γ_i's are positive tunable weights and the $f_i(T_m(t),E(t))$'s functions that each captures *one* dimension of the desired behaviour, i.e. its utility. For example, with $0,1,\ldots,j,\ldots,n$, $f_j(T_m(t),E(t))$ could be a measure of the proximity to other vehicles, f_{j+1} a measure of ride quality, f_{j+2} the representation of some traffic rule etc. For clarity, it is stressed that T_m and E are to be read as symbolic representations of the function f_i's

dependence on the trajectory evaluated and the prediction of the environment. Using this formulation, changing the weights γ_i allows a designer to affect the relative impact of any one behavioural dimension, and thereby the composition and "preference" of the resulting $\hat{J}_m$.

Implementation Considerations To ease tuning, and restrict the numerical range of $\hat{J}$, the contributing functions are designed so that $f_j(T,E) \in [-1,1]$. A positive value for a f_j thus means that some advantageous property is achieved along the dimension while a negative implies adverse impact. The functions used in the implementation of the selection system are briefly listed in Table 3.1. However, due to the restricted space available in this thesis, the construction of the different functions f_j are not detailed here. The interested reader can instead find both motivations and mathematical descriptions of the functions in Appendix C.

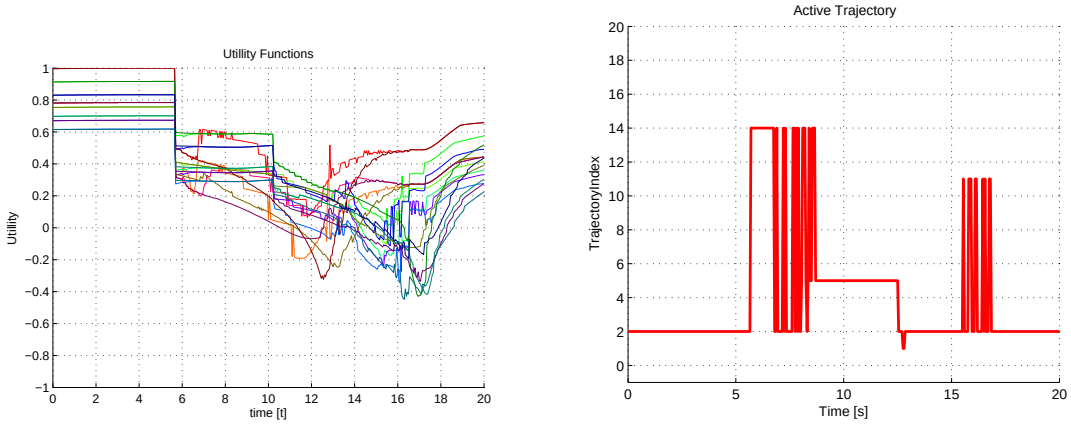
Name	Function	Description
Jerk	f_{jerk}	Penalises aggressive manoeuvres.
Maximal Acceleration	f_{acc}	Penalises high peak acceleration.
Deviation from desired lane	f_{lane}	Penalises trajectory endpoints in non desired lanes.
Deviation from lane center	f_{center}	Penalises trajectory endpoints with lateral offset to the closes lane center.
Terminal longitudinal velocity	f_{vel}	Premier trajectories with endpoint velocities close to the ACC set speed.
Proximity to obstacles	f_{prox}	Penalises trajectories that at some point comes close to any obstacle.
Stationarity	f_{stat}	Premiers completion of commenced trajectories.

Table 3.1: List of components for the utility function used in the selection process.

3.4.2 Filtering

Although attractive in its simplicity the direct usage of the weighted sum introduced in Equation 3.46 is coupled with problems. Consider a trajectory index m that at time t

holds the highest utility value $\hat{J}_m(t)$ among all calculated trajectories. Rapid changes in utility between two samples, i.e. between $\hat{J}_m(t)$ and $\hat{J}_m(t + \Delta t)$, could give that some other trajectory index k is given the highest utility value and consequently that its execution is commenced. A problem occurs when, for some reason, the evaluation at the following sample, i.e. at $t + 2\Delta t$, again gives trajectory m the highest score, and it consequently is selected and executed. In the worst case, a rapid switching back and forth between two, often similar trajectories over extended periods of time can occur, severely reducing the quality and predictability of the planning system. Unfortunately, some components of the utility value, i.e. the functions listed in Table 3.1 might include discontinuities having this effect (consider the logical *on/off* nature of the ACC target selection). Naturally, the presence of noise would severely worsen the situation, as sensor readings forms the base of the utility value components. An exemplification of the phenomenon is given in Figure 3.20 where a simple highway overtake scenario is simulated with selection based purely on Equation 3.46.



(a) (b)

Figure 3.20: Illustration of the rapid switching between trajectories in a highway overtake scenario (a) Shows the development of the utility value as introduced in Equation 3.46 over time for a set of nine trajectories, differentiated visually by color. The currently active one is taken as that with the highest value. (b) The index of the trajectory currently being executed is shown. Stationary periods implies that one trajectory is allowed to execute for an extended period of time. The rapid switching can be seen clearly between times 8-6 *s* and 15-17 *s*, where the system commences and aborts the execution of trajectories 14 and 11 repeatedly.

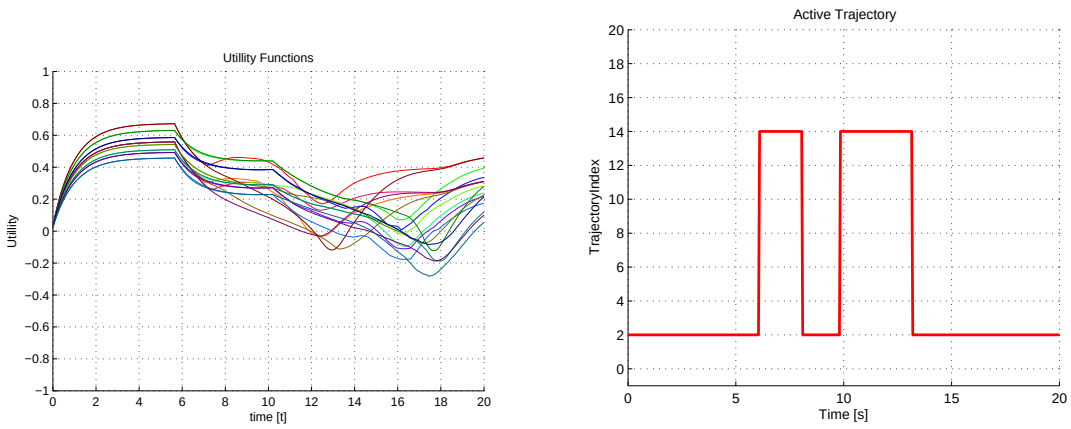
Similar consideration to exist in other fields where selection based on noisy or discontinuous data is necessary, notably in behaviour selection for mobile robots. To mitigate the issue, a special low pass filter is commonly introduced. Based on work done in this context by Wahde [24], the filtered utility $g_m(t)$ for the trajectory indexed m is set up as

$$\dot{g}_m(t)\tau + g_m(t) = \tanh(\alpha \hat{J}_m(t)) \tag{3.47}$$

and discretized using Euler integration according to

$$g_m(k+1) = g_m(k) + \left(\tanh(\alpha \hat{J}_m(k)) - g_m(k)\right) \frac{\Delta t}{\tau} \tag{3.48}$$

Here, k denotes the discrete time index while Δt is the system sampling time. The hypertangent is introduced to restrict the numerical range of the filtered value so that it always remains in the interval $(-1,1)$, provided it is initiated inside it. This is done to reduce the risk of windup like situations. The integration constant $\tau > 0$ will here govern the sensitivity, or responsiveness of the utility dynamics, where lower values lead to a more "nervous" system with rapid reaction to environmental changes. As it is lowered, the behaviour will more and more tend to a saturated version of Equation 3.46. The parameter $\alpha > 0$ affects the range of the input $\hat{J}_m(k)$ over which input variations are made visible, i.e. where the hyper tangent squashing function does not saturate. Here, lower values lead to a wider interval, higher to a more narrow. Consequently this also influences the output range of the utility function it self, however as each trajectory shares the same parameters it does not affect the relative measure dramatically. A more thorough discussion with examples and recommendations on appropriate values will be made in 4.1. Following the same selection principle where the trajectory corresponding to the highest *filtered* utility value is selected and executed, Figure 3.21 gives an exemplification of the effect of the filtering.



(a) (b)

Figure 3.21: Demonstration of the filter introduction detailed in this section. The scenario is the same simple overtake shown in Figure 3.20. However, Equation 3.48 is now used for selection rather than Equation 3.46. (a) Shows the development of the filtered utility value while (b) again visualises the index of the executed trajectory. As seen in (a), the initial values are set to zero for all trajectories.

In addition to reducing the rapid switching, the filter introduces *memory* to the selection process, which in many cases are advantageous. As an example, consider travelling with

an offset to a desired lane, e.g. in the oncoming lane on a country road. Is indeed acceptable to do this for shorter periods of time during overtakes, but highly undesirable to remain longer than necessary. The differential form in Equation 3.47 captures this by effectively allowing the negative and positive influences to "spool up" over time.

For a comparison on how the differentially formulated selection functions perform in comparison to the linear, memoryless Equation 3.46 compare Figures 3.20 and 3.21. Note here that no particular tuning is done regarding the parameters and the behaviour therefore might differ from that suitable for implementation.

3.4.3 Prediction

As mentioned in the previous sections, a prediction of the obstacle motion over some horizon is needed to efficiently evaluate and select among the generated trajectories giving that the longitudinal position and velocity of the host cannot be retrieved directly. Instead, this has to be the result of simulation based on models of the vehicle and ACC. The latter generally includes more than one mode of operation differentiating between unhindered, set speed adaptation and adaptation to some nearby obstacle. A model of the ACC must therefore by necessity also include modelling of the sensory system on which it depends as well as its target selection. The latter implies that any trajectory of the host must be based on those of the predicted obstacles. In real life, the relationship is of course bi directional as the action taken by any vehicle will affect the decision making by other road occupants. Due to this, accurate models need not only include hardware dynamics but additionally also cover driver behaviour. Even though attempts to tackle some of these problems using various probabilistic techniques indeed has been made [16], the models rapidly grows in complexity as the richness and cross coupling between the involved agents is increased. Due to the human involvement, even the best of models cannot state any guarantees or bounds and are by nature highly approximate. For this reason a fairly simple model is used in conjunction with the planning system presented in this thesis. Dependencies are considered one directional, i.e. the actions of the host does not affect the prediction of the obstacles. Further, a vehicle is assumed to stay at the same lateral position, i.e. lane, over the entire horizon. Remaining for each obstacle is thus the prediction of longitudinal position, for which a constant velocity particle model is used. Formally, the prediction of obstacle i, $i = 1,...,|O|$ is then taken as

$$x_i(\tau) = x_i(0) + v_{o_i}(0)\tau \tag{3.49}$$
$$y_i(\tau) = y_i(0) \tag{3.50}$$

The notation here is where applicable retained from previous sections and $\tau \in (0,\tau_f)$ denotes the prediction time. Arguments set to 0 are meant to be read as the initial value of the prediction, i.e. the measured values at the current sample.

The lateral motion of the host vehicle during planned highway manoeuvres is as presented in 3.2 given by a quintic polynomial $p(\tau)$ and its derivatives. For the longitudinal position, a double integrator is used. By this, the host trajectory is in continuous time given by

$$\begin{aligned} \dot{x}_h(\tau) &= v_{t_j}(\tau) && (3.51)\\ \dot{v}_{t_j} &= a_{lon}(E,\tau) && (3.52)\\ y_h(\tau) &= p(\tau) && (3.53) \end{aligned}$$

The longitudinal acceleration $a_{lon}(E,\tau)$ is considered to be the output of the ACC module. The latter is due to its hybrid nature modelled as a conditioned state feedback controller, that is

$$a(E,\tau) = \begin{cases} k_x(\Delta x_i - \Delta x_i(\Delta v_i)) + k_v \Delta v_i & \text{if obstacle } i \text{ is a target } \textbf{and } v_i < v_s \\ k_s(v_s - v_{t_j}(\tau)), & \text{if no target is present } \textbf{or} \text{ host is leaving lane} \end{cases} \tag{3.54}$$

where k_x, and k_v are the feedback gains of the relative distance and velocity to obstacle i respectively. Similarly, k_s is the set speed deviation gain, used when no obstacles are present. Further, since actuation is limited, the simulated ACC output is saturated so that

$$a_{lon}(E,\tau) = \text{sat}(a(E,\tau)), \quad \text{sat}(x) \in (a_{min}, a_{max})$$

Target selection and sensor modelling are similarly kept very simple. An obstacle is considered a target only if it is within some distance R in front of the host and in the same lane. Should more than one satisfy these criteria, the closest one is selected. It should be noted that even though the approach presented here is simple, there is in principle no limit on how detailed the used models could be. Given sufficiently fast computers, a framework like that presented here could easily be extended with, say, a bicycle model of vehicle dynamics, prediction of lateral motion etc.

4

Results

This chapter contains the main results of the thesis. Simulations from implementations of the subsystems introduced in Chapter 3 are presented. For highway driving this comprises full road scenarios including lane changes, overtakes (double lane change), planned avoidance and more complex traffic scenarios. Examples are included where the system is given a high level of autonomy and the general performance and the selection process are discussed thoroughly for each case. The demand on real time operation using limited computational resources highlighted in Chapter 1 is addressed and the subsystems performance characteristics are discussed and evaluated. Due to the simplifications made in modelling, a section on model validation for the highway trajectories is included where the simple trajectories are compared with those of a detailed model. The trajectories for turning is examined separately focusing on robustness and computational performance of the described algorithm. Results from extensive simulations are used to support claims on robustness. The gains of using an off-line fitted function to generate the optimization starting point versus the simpler heuristics prosed in literature is treated as well as its impact on solver choice. Further, arguments are put forward as to why this variation of the method is superior to the formulations given in the original papers. The optimal control extension to the problem introduced in 3.3.4 is evaluated for one cost functional, affecting the resulting trajectories.

4.1 Highway Scenarios

To validate the path planning performance of the Highway system, simulations have been made for a number of traffic scenarios. In line with the primary goals stated in 1 these have been centred around lane changes, overtakes, planned avoidance and combinations

thereof. Due to the intrinsic time dependence of the motion of a vehicle the action taken by the planner is hard to visualize on static plots, for which reason the simulation results are given as a sequence of vehicle following snapshots. Given that space is limited, four scenarios are included to demonstrate different aspect of the system behaviour. Although limited, the results are easily extendible to the general highway case. The simulation environment has been written in MATLAB specifically for this purpose.

In all shown scenarios, the arena is a three lane, uni directional road where the coordinate system is positioned so that the road bound reference coordinate (x, station) follows the center of the middle lane. The lateral coordinate (y, latitude) is taken negative below and positive above this line. The width of each lane is set to 6 m whereby the lane totals at 18. Further, the road is set to be straight during the course of the simulation. In all shown figures,the tentative trajectories are drawn at each time step and colour-coded according to a linear *relative* scale, where green is those considered attractive, e.g. with higher momentary values of corresponding selection functions, and conversely, red those deemed unattractive. The presently active one (i.e. the one coloured bright green) is at all instances drawn thicker. Other road participants are drawn together with a level curve representing their respective maximal proximity value, defined according to Equation C.4. That is, at all instances the area within these encompassing the obstacle are by the planner considered to be beyond the specified proximity limit. The sensor horizon is fixed to $120m$ during the simulations and is used as the farther limit of the snapshots.

4.1.1 Simulations

As demonstrated in 3.2.2 the inability explicitly plan longitudinal endpoints arising from usage of the ACC is handled by using different levels of maximum lateral acceleration. Due to this there will for each specified limit exist a family of trajectories leading to all lateral endpoints effectively replacing the provision of their longitudinal counterpart. However, it has been found that a single maximum lateral acceleration is sufficient for most scenarios in the highly autonomous case, the exception being situations where the host vehicle gets "trapped" in the sense that it lacks the means to plan aggressively enough to pass an obstacle. As a consequence, the cost contributions regarding $MaximumLateralAcceleration$ and $Jerk$ introduced in 3.4.1 diminishes in importance as they in principle affects the selection functions in the same direction as the *Deviation from desired lane cost*, i.e. discourages lateral movement. With the two contributions removed, the selection function reduces to

$$\dot{g}(t)\tau + g(t) = \tanh\left(\alpha\left(\gamma_{proximity} f_{proximity} + \gamma_{lane} f_{lane} + \gamma_{center} f_{center} + \gamma_{vel} f_{vel} + \gamma_{stat} f_{stat}\right)\right) \tag{4.1}$$

where the arguments of $f_x()$ has been omitted for brevity. The weights and other relevant parameter values that are shared throughout the scenarios are specified in Tables 4.1 and 4.2 and held constant over the simulations.

Table 4.1: Various simulation parameters

Parameter	Value	In Equation
D	$10m$	
k	$0.2s/m$	
c	0.5	
β	5	
a_{max}^{lon}	$1.5m/s$	
a_{min}^{lon}	$3.5m/s$	
a_{max}^{lat}	$1.5m/s$	
t_f^{lim}	$4s$	
k_v	0.5	
k_s	0.5	
k_x	0.1	
t_h	$3s$	

Table 4.2: Parameters for the selection functions

Parameter	Value	Equation
τ	1	
α	1/3	
$\gamma_{proximity}$	2	
γ_{lane}	2	
γ_{center}	0.5	
γ_{vel}	5	
γ_{stat}	4	
desired lane	*rightmost*	

4.1.1.1 Overtake

The host vehicle is approaching a slower moving vehicle from behind and is thereby forced to either reduce speed or overtake. No other road occupants are present and the obscuring vehicle is travelling at a constant speed of 65 *km/h* throughout the entire scenario. The host vehicle set speed is 90 *km/h* and it is initiated at 85 *km/h* at $x = 0$,, $y = -6$ whereas the single obstacle starts at $x = 140$ and $y = -6$. Sequential snapshots are shown in the panels of Figures 4.3 and 4.4. The corresponding state and selection functions are shown in Figures 4.1 and 4.2 respectively.

Examining the latter, from $t = 6$ and onwards one clearly sees the influence of the obstacle as the host vehicle approaches (c.f. 4.3(a)). Although all trajectories are affected, they are so to different degrees, whereby the rate of reduction for the corresponding selection function differs. Simultaneously the effects of the longitudinal velocity contribution f_{vel} increases the disparity, ultimately leading to a change in active trajectory. Note the effect of the *Stationarity* contribution as the trajectory switches are made at time instances $t \approx 6$ and $t \approx 17$, the former seen in panel 4.4(b), decreasing the possibility for chatter. Following the trajectories ending up in the rightmost lane (red hues), the effects of higher proximity score resulting from the overtake can be seen in the reduction between times $t \approx 10$ and $t \approx 17$. Roughly at $t = 12$ the proximity limit is reached (i.e. $f_{prox} = -\infty$) for two trajectories as the predicted positions of the host and obstacle violates the measure, resulting in a rapid decrease. Indication to this can be seen in the red colouring of the corresponding trajectories in panel 4.4(a). Having passed the ob-

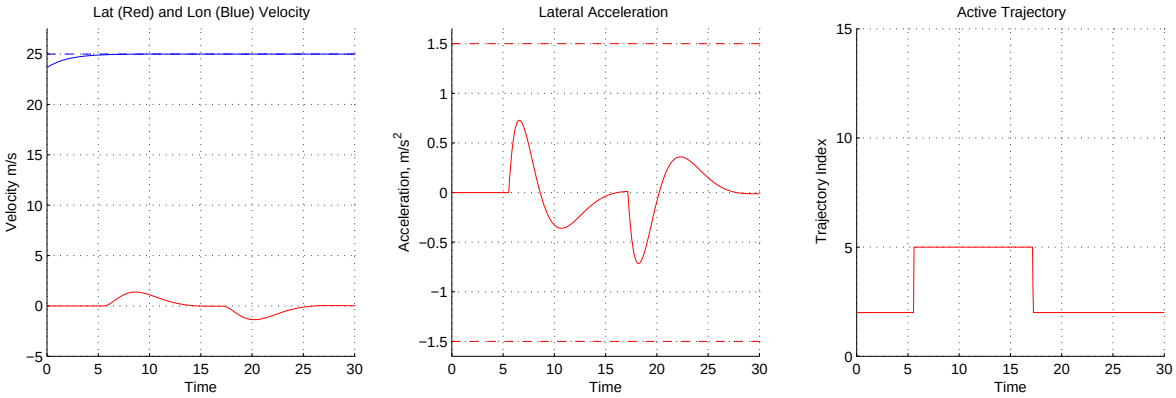


Figure 4.1: Evolution of Longitudinal and lateral velocities, lateral acceleration and the index of the active trajectory during the **Overtake** scenario. The imposed acceleration limit is shown in dashed red and the ACC set speed in dashed blue

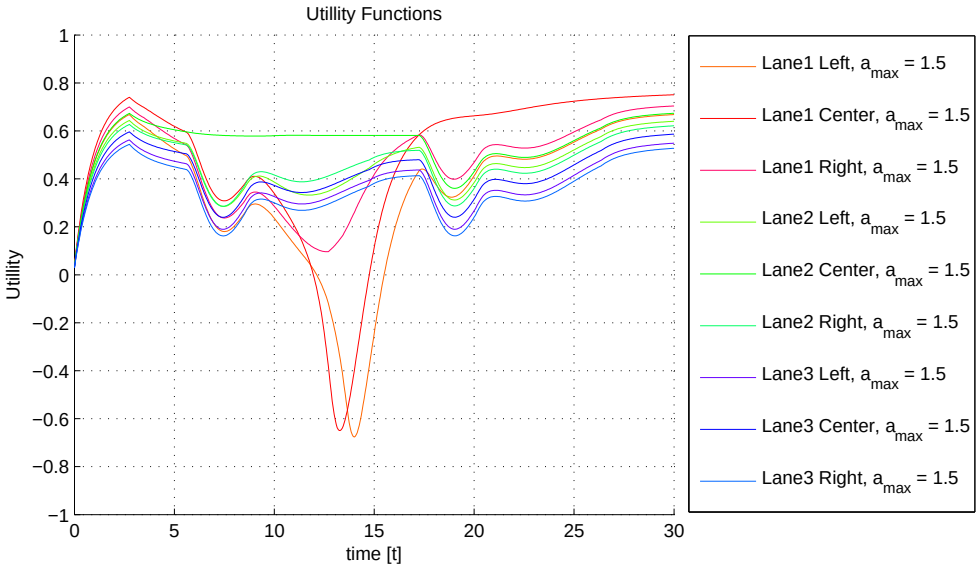
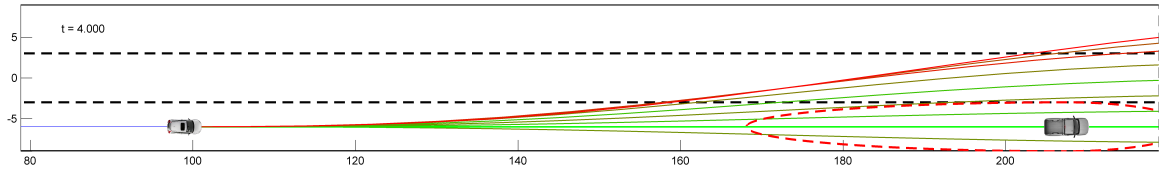
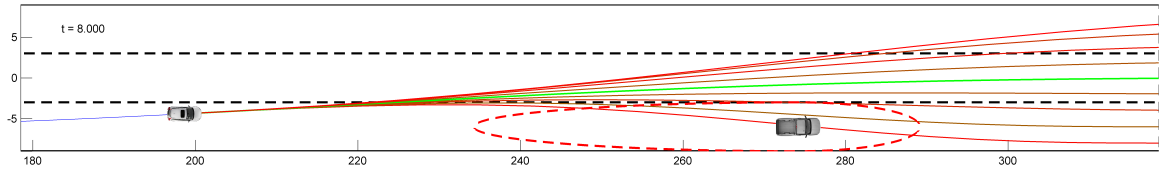


Figure 4.2: Selection functions for **Overtake** manoeuvre. Each curve corresponds to the filtered utility value of a trajectory. The corresponding lateral position of the trajectory endpoint is given in the legend, e.g. "Lane 1 Left" corresponds to a trajectory that ends in the first (rightmost) lane, offseted from the lane center to the left. The currently active trajectory is taken as that which has the highest utility.

stacle, all trajectories get decreased punishment for proximity and thereby grows. Note especially the rapid increase of the left center trajectory as f_{lane} pulls it up, ultimately resulting in a trajectory switch, c.f. panel 4.4(b).

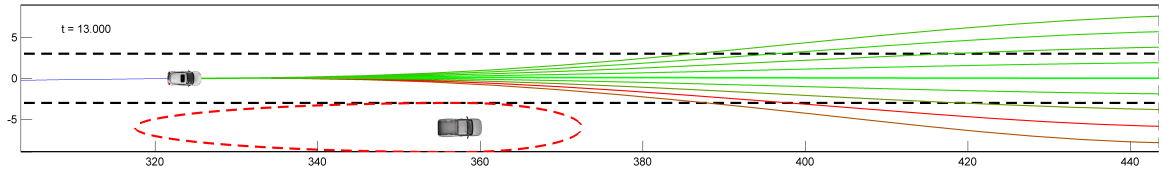


(a)

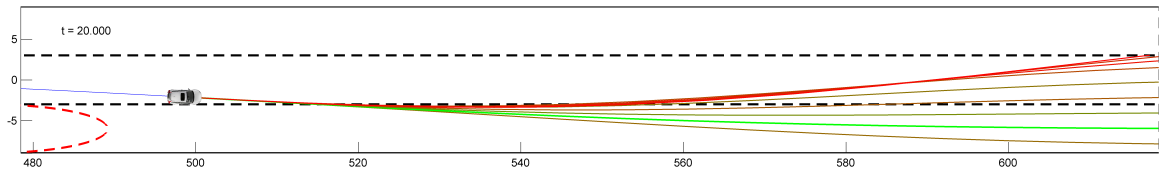


(b)

Figure 4.3: First part of Overtake. The host approaches the obscuring vehicle and selects a collision free trajectory around it. The calculated trajectories are drawn and colour-coded so that green hues shows attractive trajectories attractive, e.g. those with higher momentary values of corresponding selection functions, and conversely, red unattractive. The hues are set so that the trajectory among the currently calculated one with lowest utility are shown in pure red, the one with highest pure green and those in between according to a linear scale between the two extremes. The presently active one (i.e. the one coloured bright green) is drawn thicker. The dashed red elliptic shape around the obstacle visualizes the proximity limit as described in Appendix C



(a)



(b)

Figure 4.4: Second part of Overtake: The host passes the obscuring Vehicle and returns to its preferred lane. Colour coding according to the caption of Figure 4.3

4.1.1.2 Planned Avoidance

Similarly to the Overtake scenario, the host vehicle approaches the obstacle from behind. In this example however, the obscuring vehicle is stationary and positioned at the side of the road, only partially occupying the lane. The same initial values as in the previous scenario is used for the host, whereas the obstacle is initiated at $x = 250$,$y = -8$. Snapshots of the performed scenario is given in Figures 4.7 and 4.8 with corresponding costs and state developments shown in Figures 4.5 and 4.6 respectively. Note especially how the distance measure behaves for a stationary object, where the high relative velocity extends the forbidden zone backwards to increase safety margins. Further, the forward section of the zone is reduced accordingly, allowing an earlier, safe return to the desired lane than seen in the overtake scenario. A necessary note here is that the obstacle initially lies outside the range of the sensors but still affects the selection. This could for obvious reasons not occur in an actual scenario.

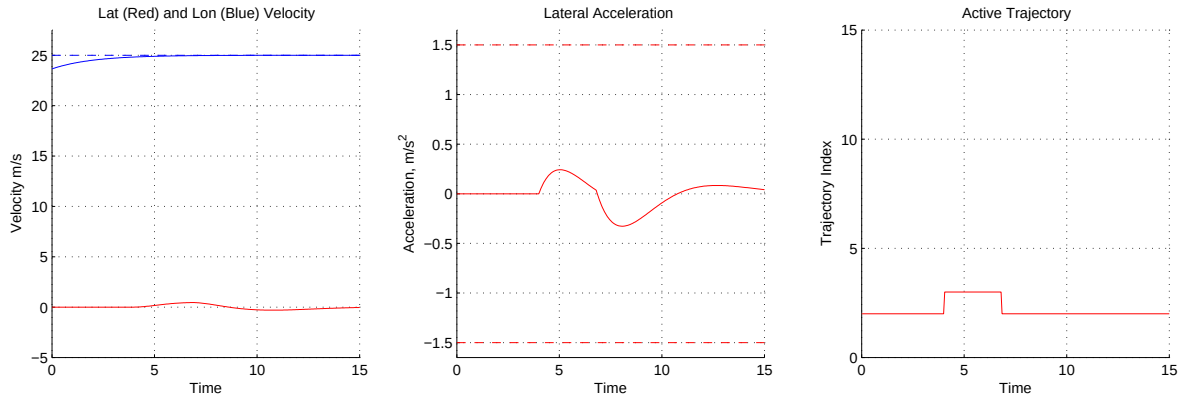


Figure 4.5: Evolution of Longitudinal and lateral velocities, lateral acceleration and the index of the active trajectory during the **Planned Avoidance** scenario. The imposed acceleration limit is shown in dashed red and the ACC set speed in dashed blue

It should again be noted how the proximity to the obstacle drives down the selection score of one trajectory, roughly after 5 seconds, as visualized in 4.7(b)

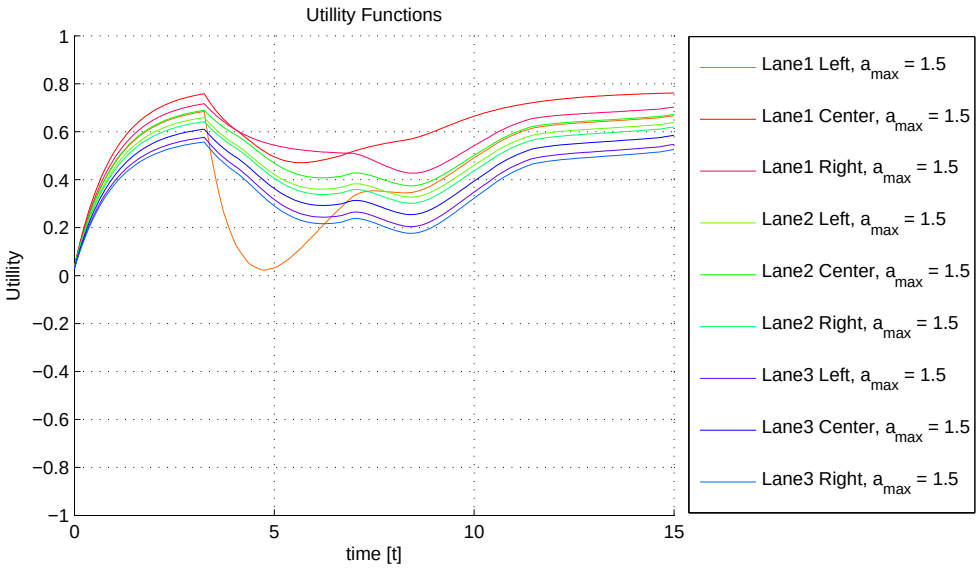
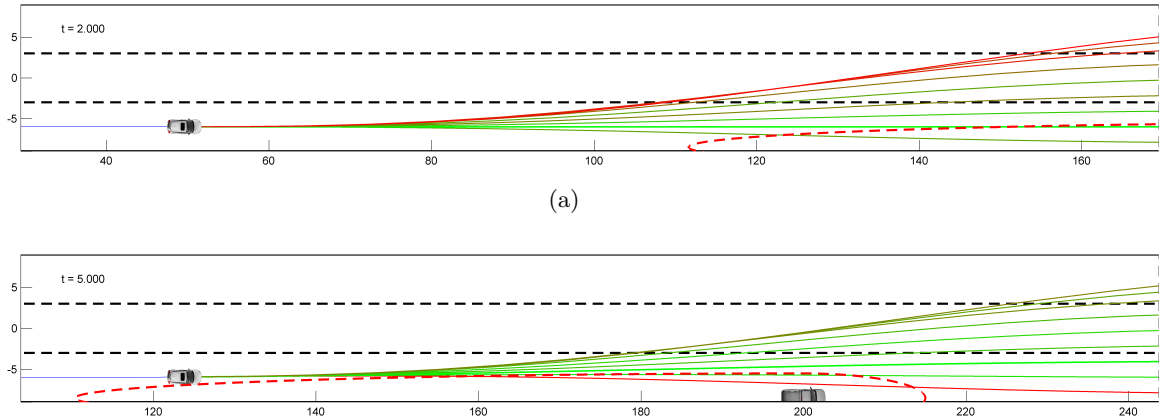


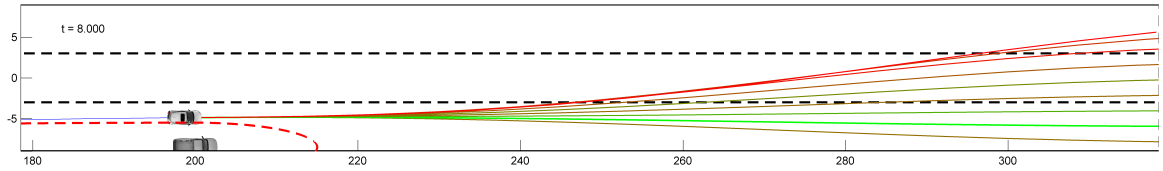
Figure 4.6: Selection functions for **Planned Avoidance**. Colour coding and naming as described in the caption of 4.2.



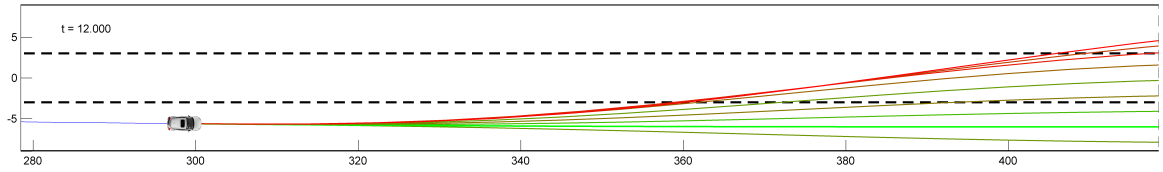
(a)

(b)

Figure 4.7: First part of Planned Avoidance: The host approach the obstacles and chooses a trajectory around it. Colour coding according to the caption of Figure 4.3



(a)



(b)

Figure 4.8: Second part of Planned Avoidance: The host passes the stationary obstacle. Colour coding according to the caption of Figure 4.3

4.1.1.3 Complex Scenario I

To validate performance over a longer period of time a more comprehensive scenario has been simulated, essentially comprised of two separate problems. In the first part, the host is presented with an obstacle configuration requiring lane changes over two lanes, in the second it needs to overtake a slower moving in a situation where a faster one is approaching from behind in the overtake lane. As in previous situations the vehicle starts at an obstacle free position in the rightmost lane with a speed of 85 *km/h*. The scenario snapshots are given in Figures 4.11, 4.12 and 4.13, with corresponding state and cost evolution shown in Figures 4.9 and 4.10

Upon detection of the first obstacle in the left most lane the vehicle commences a lane change to avoid it. However, at this point the obscuring vehicle in the middle lane are beyond the sensor horizon due to which it is not regarded as a target by the ACC. The selection therefore favours a two step manoeuvre to pass both vehicles opposed to a smooth double lane change as both options essentially have the same positive cost contribution from *Terminal Longitudinal Velocity*. The corresponding two switches occur roughly at $t = 5$ and $t = 12$ respectively. Essentially the host performs one lane change in the middle of which it detects the second obscuring vehicle., c.f. panel 4.11(b). The penalty for changing trajectory during a manoeuvre, i.e. the *Stationarity* cost, prevents it from switching until the already commenced is finished. This behaviour can be seen in topmost blue line in between 8 and 12 seconds in Figure 4.10. Having passed both vehicles, the reduction in proximity and pull towards desired lane allows the host to take the smoother two-lane change path back. During the second part of the scenario the host

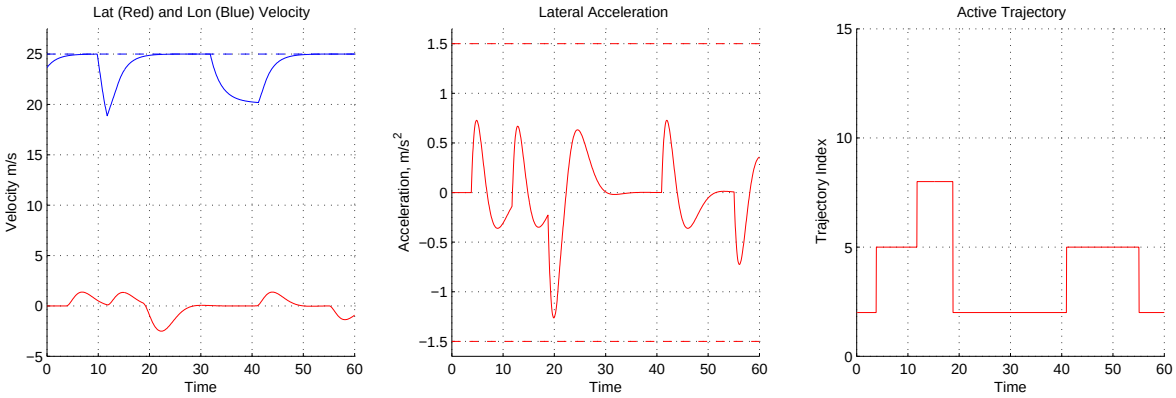


Figure 4.9: Evolution of Longitudinal and lateral velocities, lateral acceleration and the index of the active trajectory during the **Complex I** scenario. The imposed acceleration limit is shown in dashed red and the ACC set speed in dashed blue

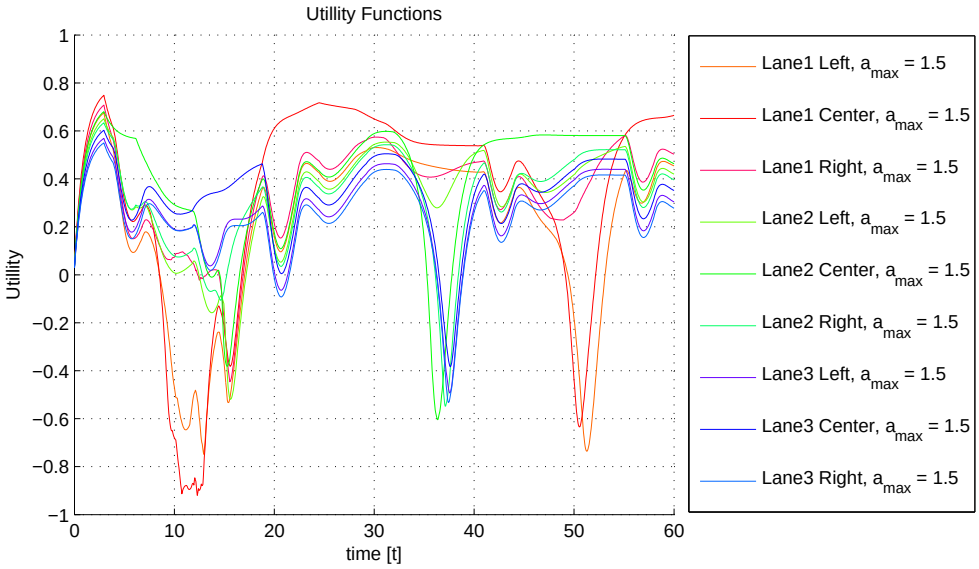
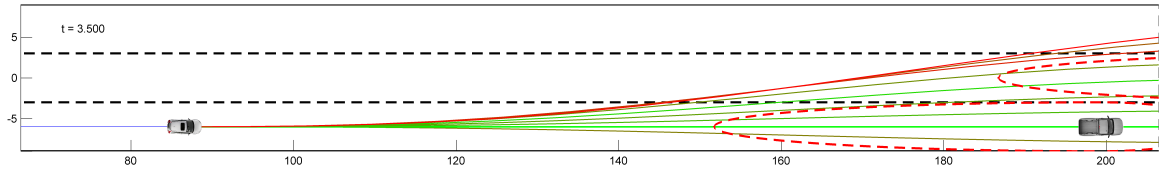
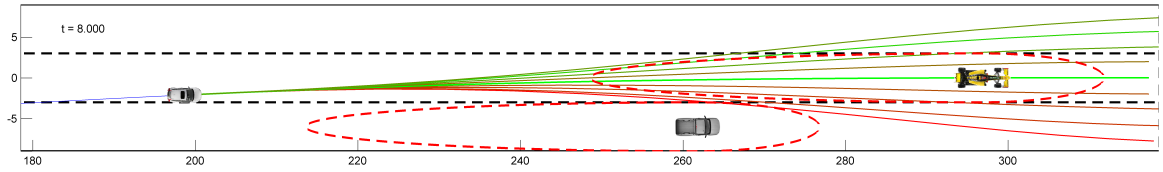


Figure 4.10: Selection functions for the **Complex I** scenario. Colour coding and naming as described in the caption of 4.2.

Note again the influence of proximity on the cost evolution: The passing to the left of the first two obstacles significantly reduces the utility of choosing trajectories to the right, seen in the decrease of red and green lines between the times 10 and 20. Similarly, when the host it self is passed by the now faster moving obstacle, the trajectories momentarily leading to left motions (green and blue hues) are decreased accordingly, c.f. Figure 4.10 between $t = 30$ and $t = 40$. Consider also the proximity isoline of the passing obstacle in panel 4.13(a) and note that the negative relative velocity extends it forward whereby unsafe actions are avoided.

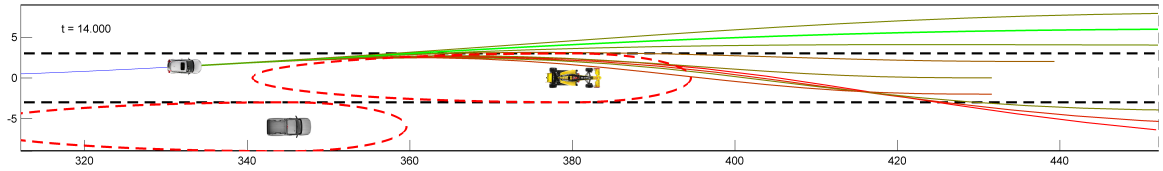


(a)

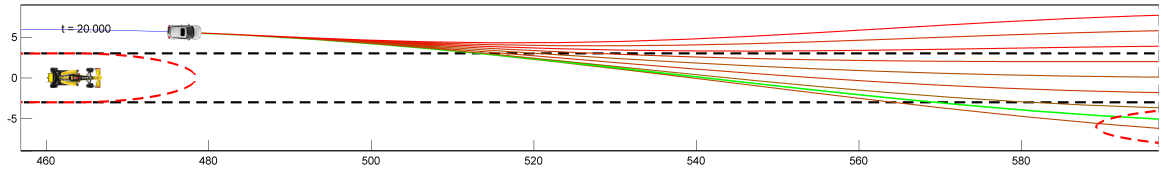


(b)

Figure 4.11: First Part Of Complex I: The host vehicle approaches the obscuring vehicles and initiates a lane change. Colour coding according to the caption of Figure 4.3



(a)



(b)

Figure 4.12: Second Part of Complex I: The host vehicle performs an additional lane change to position it self in the leftmost lane and thereafter returns to the desired rightmost one. Colour coding according to the caption of Figure 4.3

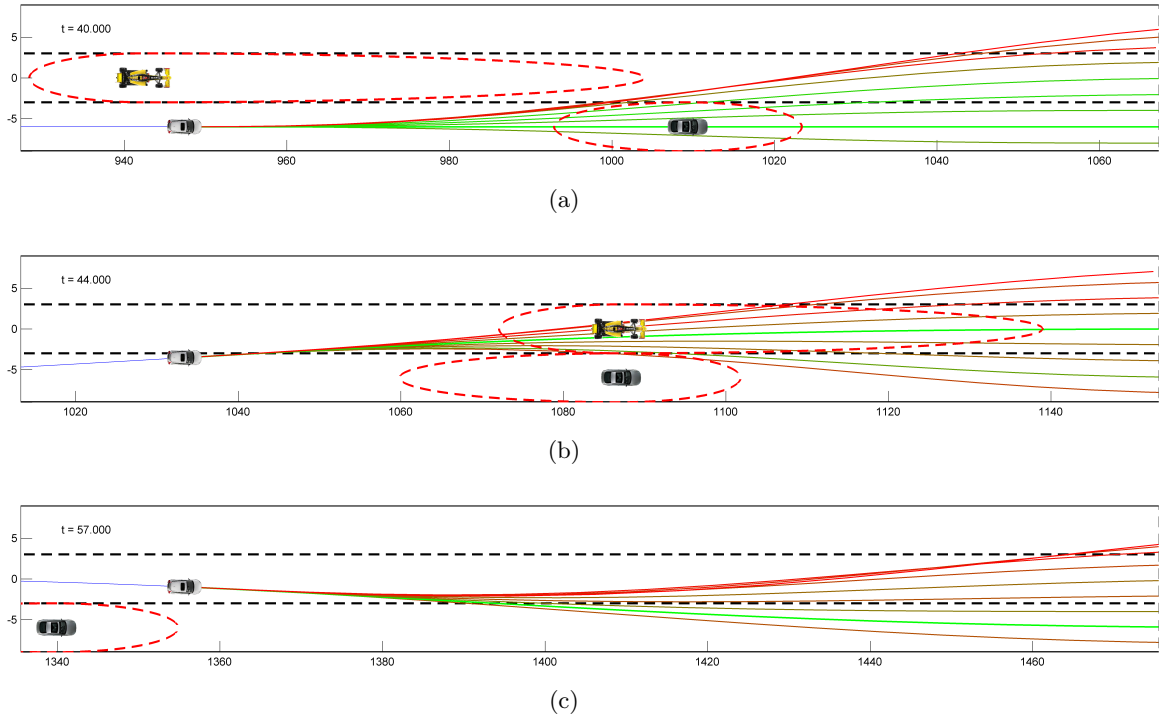


Figure 4.13: Third Part of Complex I: The previously passed vehicle increases speed and prevents an overtake. The host vehicle detects this and adapts its paths accordingly. Colour coding according to the caption of Figure 4.3

4.1.1.4 Complex Scenario II

To further investigate system behaviour in (artificially) extreme situations simulations a additional complex scenario has been performed. The purpose is to visualize some of the issues that naturally arises when planner like that described herein are used. Similar to previous examples, the host vehicle starts in the desired right lane and encounters a slower moving vehicle hindering its progress forcing it to perform a lane change. However, in this scenario the second and third lanes are blocked by other vehicles some distance ahead of the first and moving at the same speed. By this, the host is forced to take additional action. Additionally, a fourth obstacle is introduced in the rightmost lane at an additional distance in front of the first, effectively yielding a moving, "Chicane" like situation. Snapshots of the progress are provided in Figure 4.16, 4.17 and 4.18 with correspnding state and cost development shown in Figures 4.14 and 4.15.

Observations regarding the selection based on proximity can as in previous cases be made. What is interesting here however is the action taken at $t \approx$, where a lane change is performed to a offset position relative to the desired lane center. The scenario structure

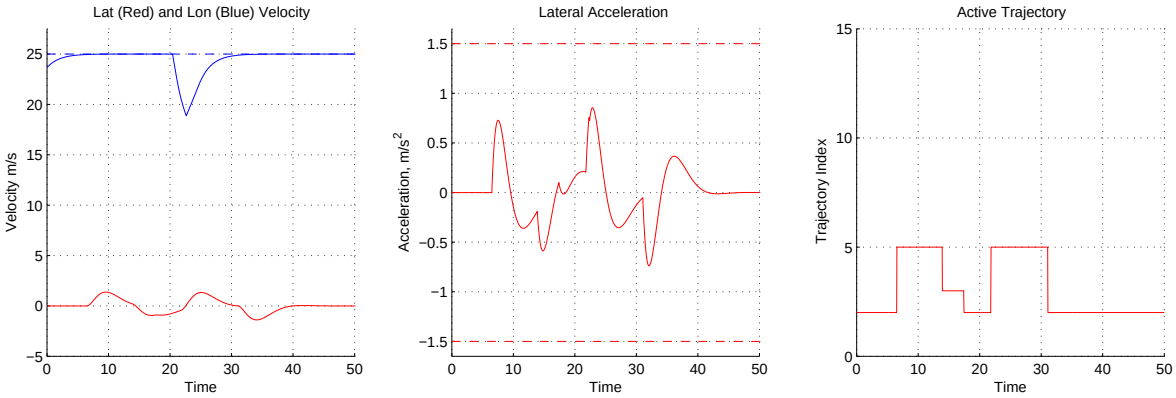


Figure 4.14: Evolution of Longitudinal and lateral velocities, lateral acceleration and the index of the active trajectory during the **Complex II** scenario. The imposed acceleration limit is shown in dashed red and the ACC set speed in dashed blue

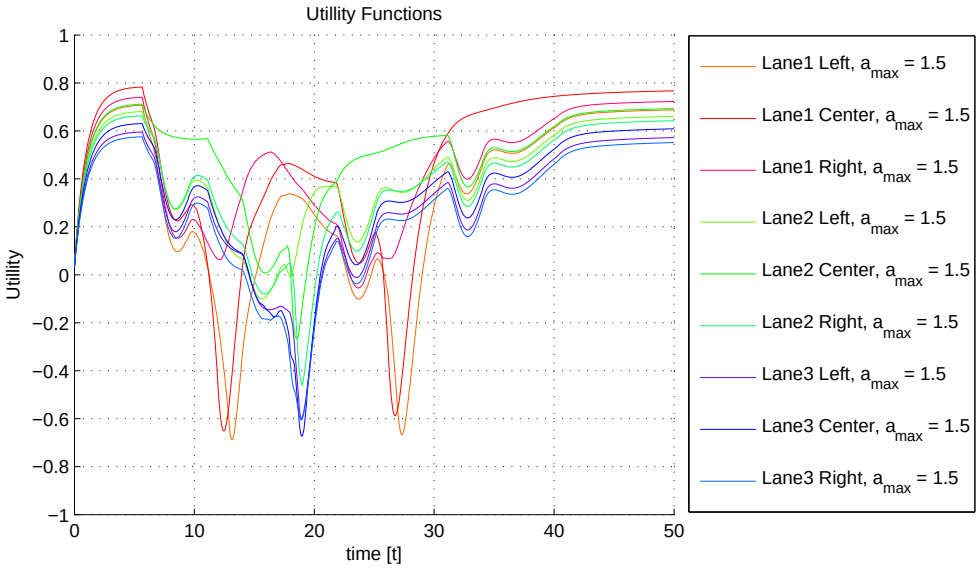
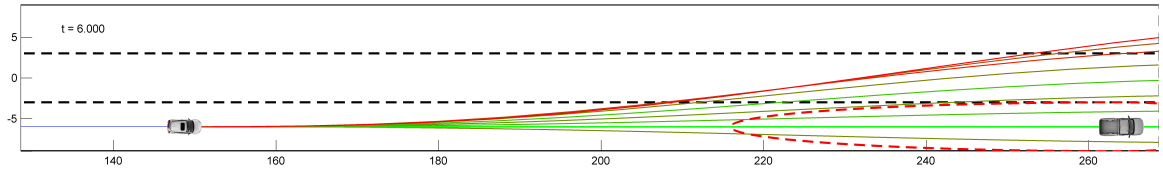


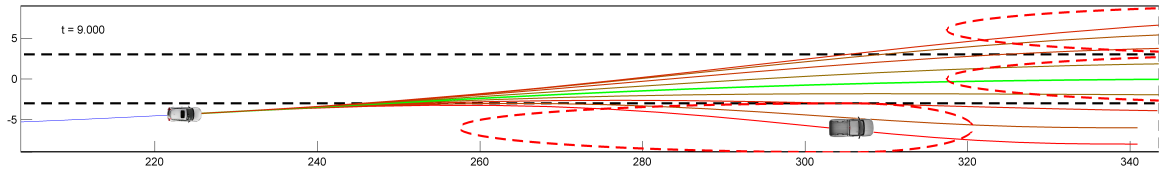
Figure 4.15: Selection functions for the **Complex II** scenario. Colour coding and naming as described in the caption of 4.2.

essentially forces this choice, as no other ways of passing between the high proximity zones exist given the set environmental parameters. However, since the relative cost increases with lane center offset, c.f. Figure C.1, the *Stationarity* contribution intended to allow completion of initiated motion is soon outweighed, resulting in a premature trajectory switch. The occurrence of situations similar to this, have been found to be symptomatic for planners based ranking and selection among a set of discrete choices. As the individual trajectories have endpoint continuity up to the second derivative, i.e. acceleration, the effects are clearly visible in its behaviour as shown in Figure 4.14. The

actual discontinuity it hereby found in the corresponding jerk .

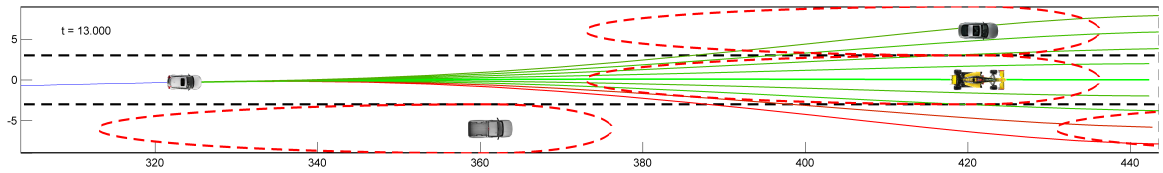


(a)

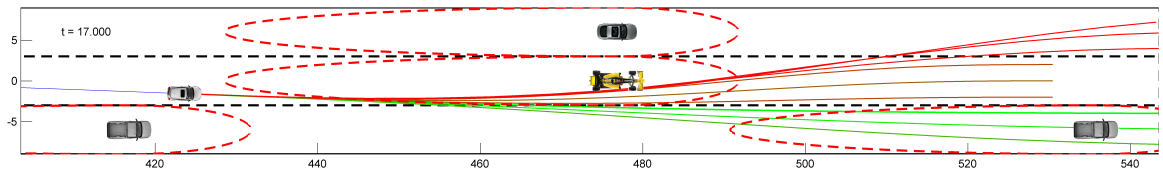


(b)

Figure 4.16: First Part Of Complex II: The host vehicle approaches the obscuring vehicles and initiates a lane change. Colour coding according to the caption of Figure 4.3



(a)



(b)

Figure 4.17: Second Part of Complex II: The host vehicle performs an additional lane change to position it self in the leftmost lane and thereafter returns to the desired rightmost one. Colour coding according to the caption of Figure 4.3

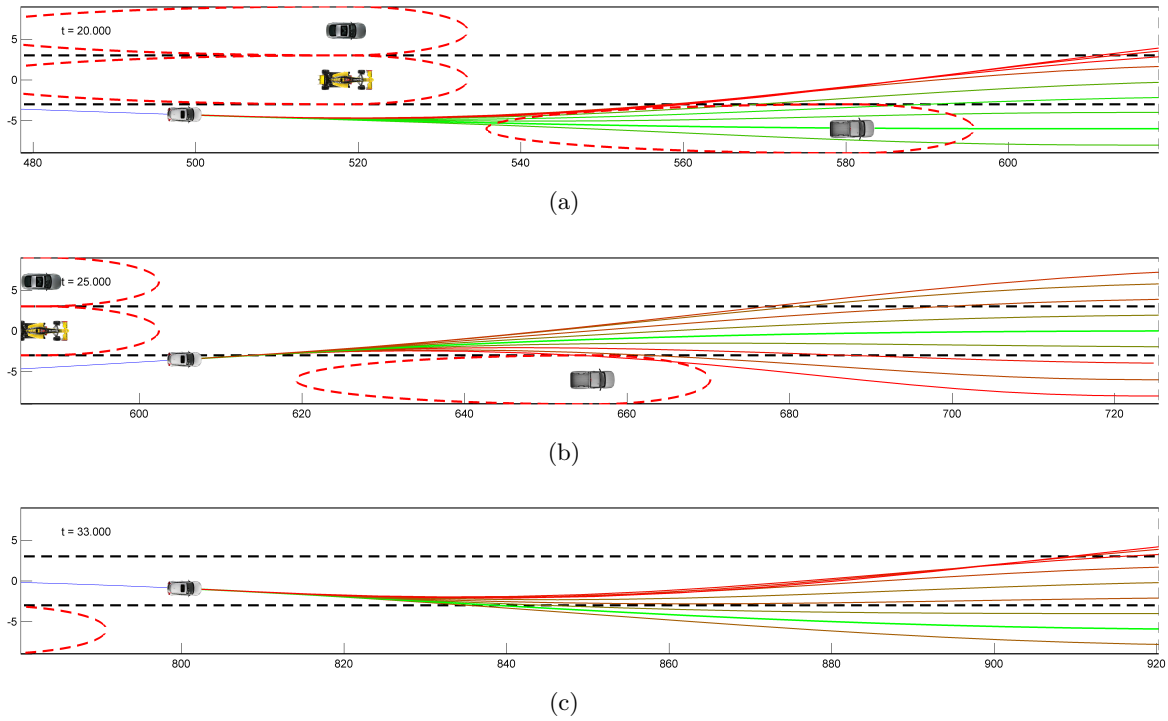


Figure 4.18: Third Part of Complex II: The host vehicle manoeuvres past the remaining obstacle and returns to the rightmost lane. Colour coding according to the caption of Figure 4.3

4.1.2 Computational Performance

As mentioned in Chapter 1 the primary constraint regarding the choice and implementation of a automotive path planner relates to computational complexity. Owing to the limited performance of present day real time platforms, the boundaries are strict even in the most forgiving case where the planner has allocated all resources, and significantly worsens as other tasks are introduced to claim processing time and memory allocation. Due to this, an elementary evaluation of the execution times of the highway planner is presented. Specifically, the change in execution times as governing parameters are varied is of interest. Even though influences from several parameters might be considered, the number of trajectories and obstacles are believed to carry the largest influence, owing primarily to the need to check every trajectory against every obstacle during both trajectory generation (due to the ACC simulation) and selection (due to proximity control).

The evaluation has been performed by logging execution times over range of parameter values, i.e. simulation set-ups, after which results are used to retrieve approximate statistic quantities. As in previous examples, the road is formed by three lanes of 6

m width. Obstacles are initiated regularly with a spacing of 80 m in the longitudinal direction while the lateral position is alternated between the rightmost and center lane, forcing the host to perform a number of consecutive lane changes. The number of trajectories are varied by alternating the number of lateral target points with lane center offset. By this each parameter increment adds 6 (two per lane) trajectories. Approximate density plots are shown in Figures 4.19 and 4.20 for variations in number of obstacles and trajectories respectively. In both cases the non varied quantity is held constant at its lowest value. Additionally, the development of the execution time mean with parameter variation is presented in Figure 4.21. The first and second panels displays the result as one parameter is varied and the other held constant and the rightmost panel, whereas the other shows the surface resulting in simultaneous variation of both. All simulations have been performed on a standard laptop (Dual Core Mobile i7(2667) @1.8Ghz using 4GB RAM) running Windows 7 HE and Matlab 2011b, without any use of parallelism.

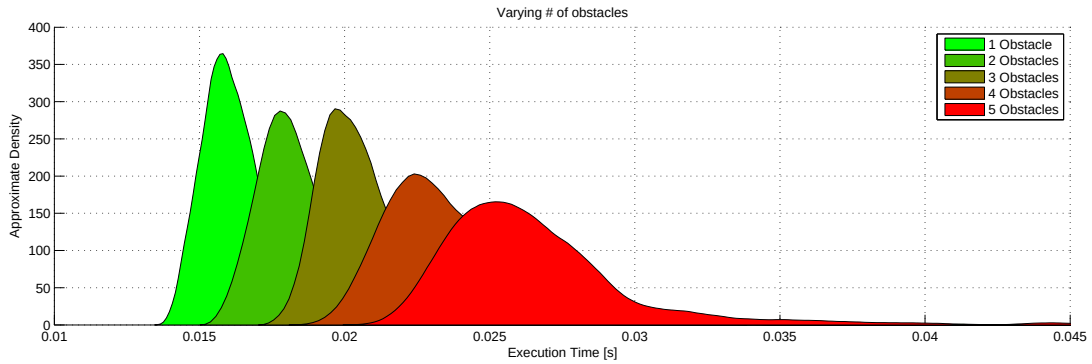


Figure 4.19: Approximate distribution of execution times for the highway planner. Comparison made on a straight road using nine trajectories. The obstacles are configured so that the host to perform zig-zag manoeuvres as it traverses the road.

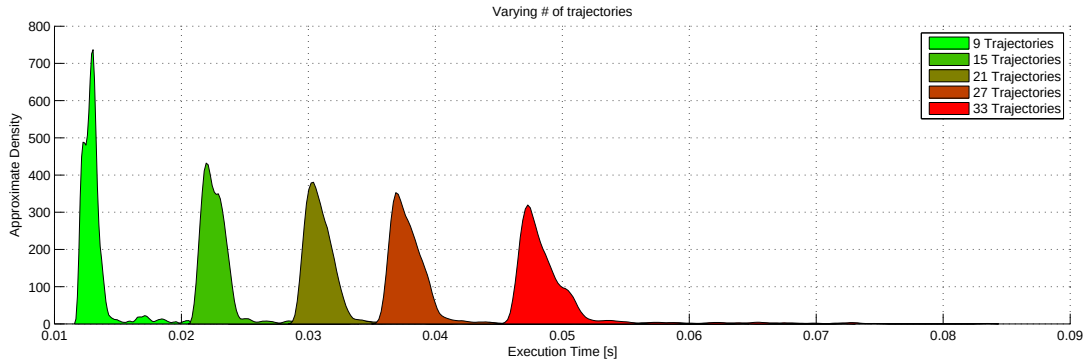


Figure 4.20: Approximate distribution of execution times for the highway planner. Comparison made on a straight road with no obstacles present. The number of trajectories are increased in numbers of six by modifying the number of lateral endpoints with a lane center offset

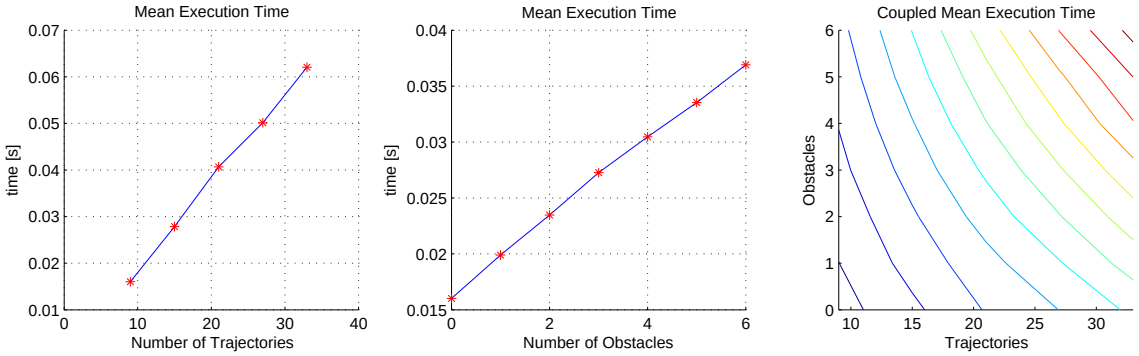


Figure 4.21: Development of mean execution time as a function of parameter variation. Note that the contour lines in the rightmost panel are interpolated between the integer values provided

It should be noted that the numerics presented herein in no way should be interpreted as absolute performance specifications, but rather as qualitative measures giving indications on how the parameters influences the needed runtime. Firstly; Matlab, beeing a high level interpreted language, runs significantly slower for any given application than an equivalent written in, say, C. Secondly; possible code optimization and other performance enhancing measures increases the disparity. Finally; as the program is executed on a general operating system its performance deteriorates further due to resource sharing with other processes running in the background.

4.1.3 Validation of Trajectories

The basic trajectories used in the Highway planner is as shown derived from very simplistic models of vehicle behavior. Even though similar ideas have been presented by several authors, e.g. in [17],[7] or [15], and been widely used in the path planning context, very limited effort have been spent examining model fidelity. Despite this, the matter is of great importance, primarily since it can affect the actual feasibility of planned trajectories, i.e. in the worst case render them in followable. Generally, any measure of optimality achieved in the derivation of the trajectories is also likely to be lost if the model is in poor agreement with the actual vehicle. Additionally, and directly related to the method examined via Algorithm 5, any constraints imposed on the simplified model suffer the risk of violation.

Due to this, the evolution of the vehicle dynamics as predicted by the simplistic model need to be compared with those of the actual vehicle. Lacking an appropriate test set-up, this is not done directly but inferred from comparisons of the simplified results to those of a detailed model. Provided that states remain in the high fidelity state spaces regions of the latter this can be said to be accurate enough.

Since the trajectories presented in 3.2 and 3.3 are based on Optimal Control formulations, a natural idea is to solve the same problem subjected to more complex dynamics. That is, the particle dynamics in Equation 3.3 should be replaced with, say, Equation E.4 and the solution used as a basis for comparison. However, there are difficulties in this relating to minimization of a third derivative. Commonly, the more detailed models are non-linear and expressed as first or second order differential equations that when differentiated one or two times both tend to be more non-linear and include first and second derivatives of the input. Combined, these formulations present problems severe enough to prevent commercial optimal control solvers like TomLAB/PROPT and BOCOP to find accurate solutions.

For these reasons, the chosen validation method instead makes the comparison between the behaviour of the simplified model and that of a more advanced one as it is *forced* to follow the physical (x,y,φ) trajectory of the former. Formally, the evolution of the advanced model will then follow

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u}) \tag{4.2}$$
$$x_x(t) = g_1(t,\mathbf{x}) \tag{4.3}$$
$$x_y(t) = g_2(t,\mathbf{x}) \tag{4.4}$$
$$x_\varphi(t) = g_3(t,\mathbf{x}) \tag{4.5}$$
$$\mathbf{x}(0) = \mathbf{x}_0 \tag{4.6}$$

where $g_i(t,\mathbf{x})$ describes the planned trajectory and x_x, x_y, x_φ the center of gravity position and direction states respectively. The restriction of the system to the predefined trajectory introduces equalities and thereby transforms the differential equation of the more advanced model to a *Differential Algebraic Equation* (DAE). Several solvers exists that handles DAE's but for reasons of availability the comparative calculations presented below has been done with the *Modelica* software DYMOLA.

Comparative Measures The result of any model comparison should give some information in what range of certain parameters, or more generally under what conditions, the model is qualitatively *"good"* or *"bad"*. Refraining from a precise definition at this point and assuming that a specific set of parameters yields feasible trajectories (with respect to the simple model), the ranges are sought after where the result can be transferred to the actual vehicle. This can then be inferred though the comparison with the more advanced model, provided the states are confined to the high fidelity regions in the state space of the latter, e.g. small enough slip angles for linear tyre characteristics or high enough speeds.

In accordance with the focus on comfort limits, as mentioned in 2.3 the evolution of the lateral jerk and acceleration are of interest. However as the latter is the derivative of the former, their respective discrepancies will qualitatively be of the same character and it is therefore deemed sufficient to focus the study on acceleration. Two measures are

used to capture different aspect of the modelfidellity. Firstly, a regular RMS value is taken over the arclength of the simple trajectory, intended to capture the general model fidelity. Formally, the RMS is defined as

$$E_{RMS} = \sqrt{\frac{1}{s_f}\int_0^{s_f} (a^A(s) - a^S(s))^2 ds} \tag{4.7}$$

where $a^A(s)$ and $a^S(s)$ are the lateral acceleration at position s for the advanced and simplified model respectively. In practice however, a discrete version is used.

The second method of verification is intended to capture the simple models ability to provide usefull bounds on the actual state evolution of the vehicle. Motivated by the direct usage of maximum lateral acceleration in Algorithm 5 the measure is therefore taken as

$$E_{max} = \max_s(|a^S(s)|) - \max_s(|a^A(s)|) \tag{4.8}$$

By this, underestimates in terms of maximal lateral acceleration will yield $E_{max} < 0$ whereas overestimates that $E_{max} > 0$.

Validation Setup The equalities enforcing the physical trajectory of the simplified model are derived from Equation 3.13 and implemented as

$$\tau = x/V_{lon} \tag{4.9}$$

$$y = a + b\tau + c\tau^2 + d\tau^3 + e\tau^4 + f\tau^5 \tag{4.10}$$

$$a^S_{lat} = 2c + 6d\tau + 12e\tau^2 + 20f\tau^3 \tag{4.11}$$

Here, (x,y) are the global position of the vehicle model while the polynomial coefficients a,b,c,d,e,f are those presented in Equation 3.7 and a^S_{lat} the lateral acceleration of the particle model as taken from Equation 3.4. Note that τ in this case not denotes time, but should be regarded as a trajectory parameter. The latter arises as the polynomial trajectory essentially is one dimensional and the adaptation to two dimensions, i.e. Equation 3.13, only will be valid in its original form for holonomic systems. In other words, the longitudinal velocity can no longer be said to be purely in the x direction, as the advance vehicle model includes orientation. The enforcement is schematically described in Figure 4.22.

The fidelity measures E_{RMS} and E_{max} can now be considered function of the scenario parameters, i.e. boundary conditions and final time of the simple model trajectory. Using a standard lane change trajectory, c.f. Figure 3.8, assuming stationary trajectory endpoints (e.g. zero lateral velocity and acceleration) and fixing the lateral target, the final time is left to be varied. However, as the longitudinal velocity of the vehicle is disregarded in the simplified model but highly relevant in an actual vehicle the parameter variations are not taken over the final time directly. Instead, each test case is defined by the longitudinal velocity and distance over which it is made. Since $t_f = x_{max}/v_{lon}$

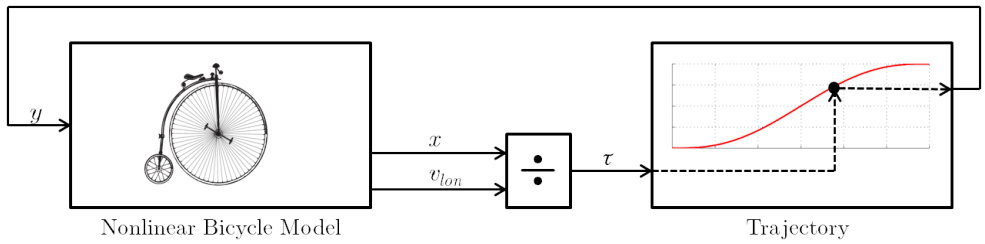


Figure 4.22: Schematic visualization of the trajectory edforcement in the DAE formulation

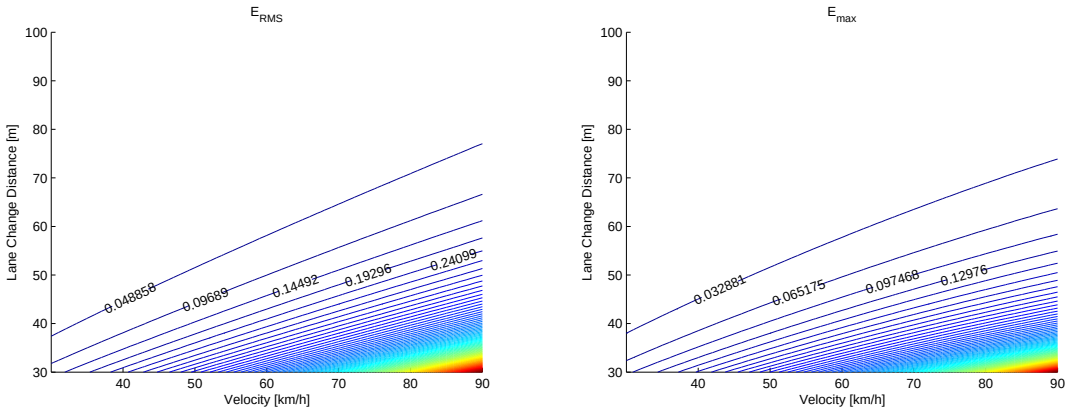
this gives the rule on how the trajectory is defined. Considering x_{max} and v_{lon} as the defining values, the test case character varies from aggressive (low x_{max}, high v_{lon}) to nice (high x_{max}, low v_{lon}) lane change manoeuvres over some envelope $x_a \leq x_{max} \leq x_b,\ v_a \leq v_{lon} \leq v_b$.

Simulations Initiated at $(x,y) = (0,0)$ and using the fixed lateral target offset of y_{max}, roughly 4300 simulations have been done on the parameter grid 30 $m \leq x_{max} \leq$ 100 m, 30 $km/h \leq v_{lon} \leq$ 90 km/hb, regularly spaced with 1 m and 1 km/h in the distance and velocity dimensions respectively. The non linear bicycle model used is for each case initialized at rest in all states save the longitudinal velocity, which is set to the current v_{lon} value. Further, no driving force is applied and rolling resistance is neglected as it is believed to have a marginal impact on the considered states. Other parameter values used are retrieved from [9] and summarized in Table 4.3.

Parameter	Value	Description
m	2000 kg	mass
J	3000 $kg\ m^2$	Yaw inertia
l_r	1.5 m	Distance to rear wheel from Cog.
l_f	1.3	Distance to front wheel from Cog.
C_f	$0.95l_r/(l_f + l_r)80000$	Front tyre stiffness
C_r	$1.05l_f/(l_f + l_r)80000$	Rear tyre stiffness

Table 4.3: Parameter values used when implementing the nonlinear model described in Appendix E in Dymola.

Primary results are given in Figure 4.23 as the contours of E_{max} and E_{RMS}. An alternative visualization of the results are given in Figure 4.24. Here the manoeuvre distance is kept constant while the longitudinal velocity is varied from low to high. To enable easy comparison the resulting acceleration profiles are normalized according to



(a) (b)

Figure 4.23: Map of E_{RMS} (a) and E_{max} (b) over a range of scenario conditions

$$\tau_N = \frac{t}{\tau_f} \tag{4.12}$$

$$a_N^S(\tau_N) = \frac{x^S(\tau_N \tau_f)}{\max_\tau(a^S(\tau))} \tag{4.13}$$

$$a_N^A(\tau_N) = \frac{x^A(\tau_N \tau_f)}{\max_\tau(a^S(\tau))} \tag{4.14}$$

so that $a_N^S, a_N^N, \tau_N \in (0,1)$ and all simple model trajectories (references) coincide.

As the advanced model used itself is a simplification of the vehicle dynamics, the validity of the comparison is highly dependent on its accuracy and care must be taken when analysing the results. Even though a number of considerations can be made, the primary limiting component in the model to vehicle fidelity for the advance case used is taken as the tyre modelling. In richer vehicle models, the inherent non-linearity of tyre characteristics is represented with more realistic descriptions, including saturations in the slip angle to tyre force relationship [25]. The model used for comparison neglects this and utilizes a linear tyre model due to which the simulated vehicle will be able to perform more aggressive manoeuvres than would be possible in an actual vehicle. For these reasons, the results given in Figure 4.25 are trimmed using an isoline of the following relationship, when taken as a function of x_{max} and v_{lon}:

$$\alpha_{max} = \max\left(\max_{s\in(0,s_f)}(\alpha_1(s)), \max_{s\in(0,s_f)}(\alpha_1(s))\right) \tag{4.15}$$

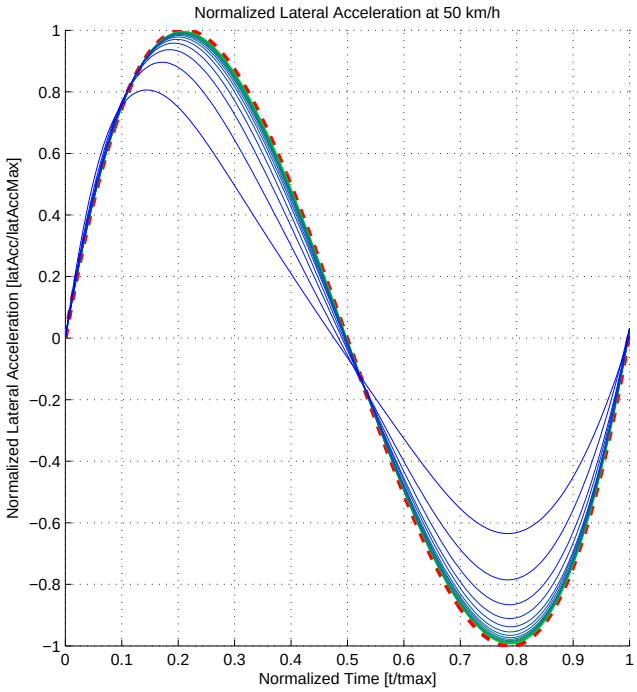
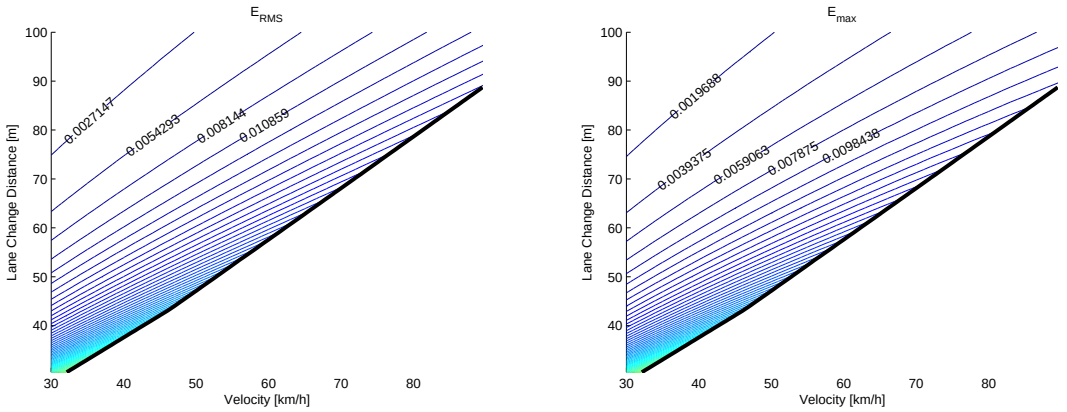


Figure 4.24: Differences between acceleration of simplified and advanced model for a lane change manoeuvre with a lateral displacement of 5 m over 50 m. The speed at which the manoeuvre is performed is varied from 30 km/h (green) to 90 km/h (blue) in steps of 5 km/h. The simplified model reference is shown in dashed red.

where α_1, α_2 denote the front and rear slip angles according to Figure E.2. That is, all sections of Figure 4.23 with corresponding $\alpha_{max} > \alpha_{lim}$ are considered in poor agreement with the vehicle and is therefore discarded. From [9], $\alpha_{lim} = 5$ is set as an appropriate limit.

4.2 Trajectories for Turning

Similar to the highway case the trajectory generator for three dimensional turning manoeuvres are validated through simulations. No implementation has been made into the general selection system whereby obstacle avoidance and other external criteria are omitted. Due to the implicit, numerical nature of both the actual algorithm and the initial guess generator as well as the complex form assumed by both (e.g. generalized Fresnel integrals, numerical integration and neural networks) no formal statements can be given regarding the general behaviour. Instead, results are derived from massive simulations and statistical measures, from which both qualitative and quantitative statements can



(a) E_{RMS} (b) E_{max}

Figure 4.25: The resulting error contours of E_{RMS} and E_{max} restricted by the 5 degree slip angle limit. The white portion thereby constitutes the region where the advance model fidelity is deemed to be lacking.

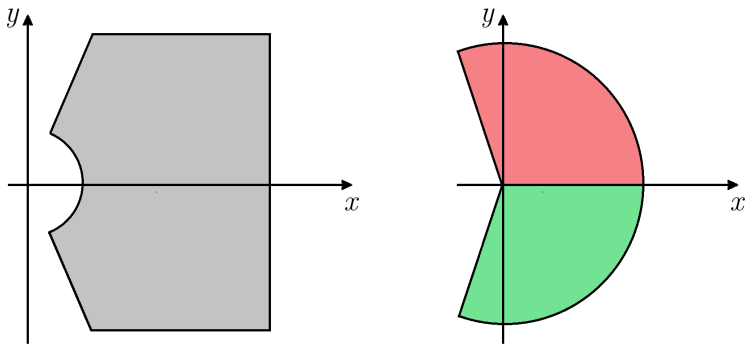
be made regarding robustness and performance.

4.2.1 Feasible Paths

The evaluations are made by simulating the trajectory generation process repeatedly with boundary conditions drawn randomly from a uniform distribution over the envelope

$$
\begin{aligned}
(x_1,y_1) \quad &: \quad \left(\sqrt{x_1^2 + y_1^2} \geq R_{min} \wedge -\alpha x_1 \leq y_1 \leq \alpha x_1 \wedge -y_{max} \leq y_1 \leq y_{max} \wedge 0 \leq x_1 \leq x_{max} \right) \\
\varphi_1 \quad &: \quad \begin{cases} 0 \leq \varphi_1 \leq \varphi_{max} & \text{if } y > 0 \\ -\varphi_{max} \leq \varphi_1 \leq 0 & \text{otherwise} \end{cases} \\
\kappa_1 \quad &: \quad \kappa_1 \in (-\kappa_{max}, \kappa_{max})
\end{aligned}
$$

as visualized in Figure 4.26. The numerical parameters used are $R_{min} = 5m$, $y_{max} = 15m$, $x_{max} = 30m$, $\varphi_{max} = 110°$ and $\kappa_{max} = 0.21/m$. The envelope are intended to cover standard turning manoeuvres occurring in normal operations, due to which terminal angles above 90° and down to 0° have been included. The conditionings and other non standard measures taken is included to exclude unnecessary cases which never would arise during normal operation, e.g. a repositioning to $x = 10$ $y = 5$ with terminal heading $\varphi_1 = -90°$ or $x_1 = 0$, $y_1 = 1$ with $\varphi_1 = 100°$. Specifically, the conditioning on the sign of y_1 prevents turns to the left ($y_1 > 0$) to have terminal headings pointing to the right and vice versa. The terminal curvature κ_1 are similarly to the lateral acceleration



(a) Spatial (b) Heading

Figure 4.26: Schematic illustration of the evaluation envelope. (a) Gives the shape of the locations of the spatial endpoints. (b) Is intended to visualize the allowed angular endpoints for left(red) and right (green) turns.

in the highway simulations set to zero for all cases, as lateral stationarity is assumed to be desired at the endpoint. Note however that this in no sense affects the general workings of the algorithm and that it is very simple to add.

It is emphasized that the statistics presented for the simulations only covers the retrieval of the governing parameters (d,s_f) and not the computation of the trajectory that is plotted. The latter constitutes numerical integration of $x(s)$ and $y(s)$ from Equation 3.17. This is by all means not unimportant, but can generally be implemented for very rapid execution by usage of the Composite Simpson Quadrature for arbitrary many points in $(0,s_f)$. In general, the systems retrieves the trajectories for the desired use case with ease, with typical results exemplified in Figure 4.27.

4.2.1.1 Generation of the Initial Guess Function

This section contains information on how the neural network that provides the initial guess for the numerical solution of Equation 3.29 is constructed. Details are given on how the training data is generated, what network topology is used, and the quality of the fitted function.

The data set used for training the neural network is drawn from an envelope of initial conditions similar to that presented in the previous section, albeit with different limiting values. To increase the coverage on the set, the envelope is made wider by letting $R_{min} = 3m$, $y_{max} = 25m$, $x_{max} = 50m$, $\varphi_{max} = 110°$ and $\kappa_{max} = 0.3$. α is set to "infinity", i.e. the linear limitation is removed, allowing all $y_1 \in (-y_{max},y_{max})$. From the uniform distribution on this envelope 10000 boundary conditions are drawn and solved for the correct minima at each iteration of Algorithm 1. The solutions are found

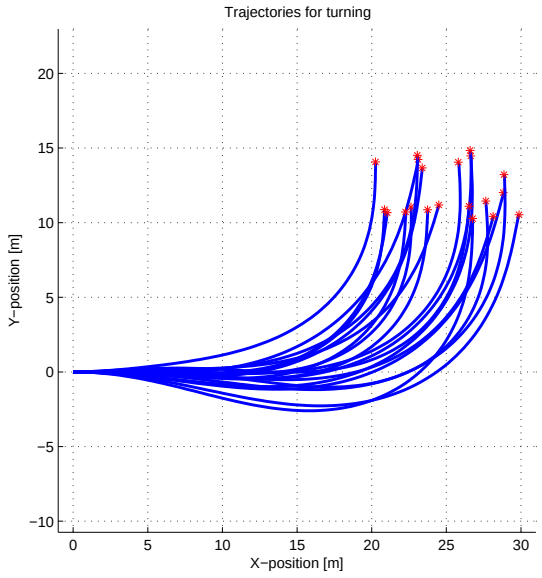


Figure 4.27: Example on system behaviour for trajectories generated over $10 \leq y_1 \leq 15,\ 20 \leq x_1 \leq 30,\ 100 \leq \varphi_1 \leq 90,\ -0.05 \leq \kappa_0 \leq 0.05$. The Plotted trajectories are formed from $x(s)$ and $y(s)$ in Equation 3.17 by Composite Simpson Quadratures

using Matlabs built in multidimensional rootfinding tool *fsolve* which runs a robust implementation of the Levenberg Marquardt Algorithm. The numerical integration is handled via Matlabs built in, adaptive interval length Composite Simpson Quadrature function *quad*, which continuously changes and recalculates the integral until a desired error bound is reached.

The neural network function is constructed using the Neural Network toolbox in Matlab, where easy to handle tools exist for function fitting. The input and target data are both brought into the interval $(-1,1)$ using min-max scaling, i.e. given some set of data D all entries are reshaped according to

$$x_s = \frac{2}{x_{max} - x_{min}} (x - x_{min}) - 1$$

before usage. Here, $x_s \in (-1,1)$ denotes the scaled variable, x the original and $x_{max},\ x_{min}$ the extreme values over the used data set. The weights and biases are adapted using the Levenberg Marquardt backpropagation algorithm, which essentially is an adaptation of what is described in 3.3 to the backpropagation context. Further, the training is done using *holdout validation*, where 15% of the original data set is randomly selected and set aside for validation purposes, and another 15 % for testing. In short, the training proceeds by adapting the network parameters based on the *training* data whereby its performance is increased. The process continues until the validation set performance stops increasing, by which the risk of fitting the function too tightly to the training data

is reduced. Further descriptions on backpropagation, holdout validation and other neural network specific details are refrained from here, for more information see for example [23]. By using the Matlab tool, the network topology is restricted to one hidden layer with hypertangent activation functions, and an output layer with linear, i.e. unsaturated, counterparts. For this application, a function using 30 hidden units have shown good performance and is therefore used. The training starting point, or the initial values of the network parameters, are set randomly whereby some repetitions generally are needed before the performance is acceptable.

Using this set up, Algorithm 1 needs three iterations before a success ratio $r > 95\%$ is reached. The fraction for which failure to converge to the correct minima occurs lies in general in the "fringe" of the envelope consisting of aggressive bends with large angles towards target positions close to the origin. Even though full coverage is desirable, it is not pursued since the intended use case of the function lies well within its safe region. The actual function to be used is retrieved as the network trained on the data set given from the last iteration, and is implemented as a stand alone m-file which the optimization procedure calls upon entry. As a measure of performance, the residual error ε of the network is formed as

$$\varepsilon = \langle ||\mathbf{p}_{NN} - \mathbf{p}_{F^*}|| \rangle$$

where $\mathbf{p}_{NN}$, $\mathbf{p}_{F^*}$ denotes the fitted and original outputs respectively. Note here that the angular brackets denotes the average value over the set used, i.e. training or validation. The development of this measure over the training process in the last iteration is shown in Figure 4.28, where the values over the training, validation and test sets are presented. Note that one *Epoch* is one pass over the training set consisting of roughly 7000 input-target pairs.

It should be noted that the same approach initially was applied for the case where *both* boundary conditions on curvature was set to zero for which significantly better results was achieved using only 15 hidden units.

4.2.1.2 Robustness of the algorithm in the merely feasible case

To assert the quality of the neural network based function for generation of initial guesses, 1000 runs have been made with boundary conditions randomly drawn from the envelope presented in 4.2.1, thereby including both "nice" and relatively hard manoeuvres. Further, the number of function evaluations for the numerical integration given in Equation D.13 is set to 20. Lower numbers are indeed possible for nicer manoeuvres, i.e. where no sharp turns and thereby smoother integrands are present, but the parameter is kept at a higher value to promote accuracy also for more extreme cases. The results are used to form approximations of the statistical properties of the function and its fidelity with respect to the actual mapping from boundary conditions to the correct minima. By this, the performance is calculated and presented as the approximate density function in the

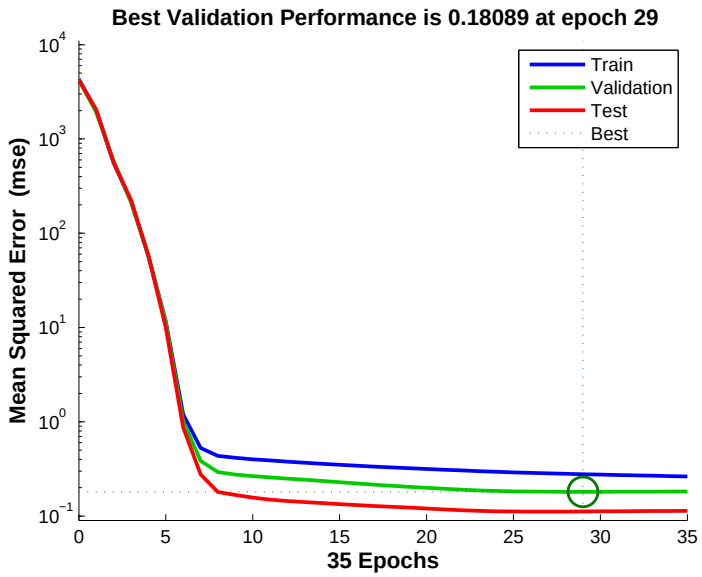


Figure 4.28: Development of the training process in the third iteration of Algorithm 1. The residual error for the training, validation and test sets are shown in blue, green and red respectively. The network used is the one giving the lowest validation set error, marked in the figure with a circle

quality measure, taken as the signed deviation from the retrieved minima in the two scaled parameters s_{fs} and d_s. The results are presented in Figures 4.29 and 4.30. In both cases the performance on the same set of boundary conditions using the heuristic based initial guess proposed by Kelly and Nagy [3] for comparison.

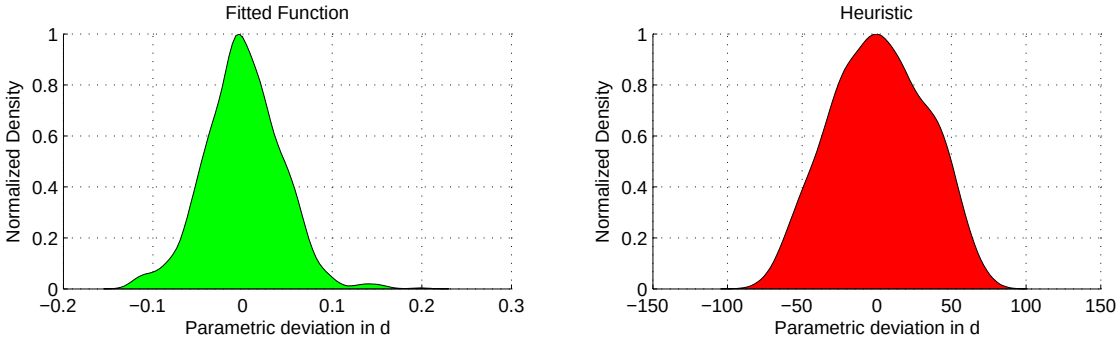


Figure 4.29: Parametric distances between initial guess and minima in the d dimension

It is highlighted that *no* failures to converge to the correct minima was registered for the neural network based implementation, whereas a number of occurrences where found for the heuristic based one. In general, the fitted function manages to place the starting point extremely close to the desired minima, removing all possibilities of erroneous convergence or divergence. An exemplification of this for three different boundary conditions is given

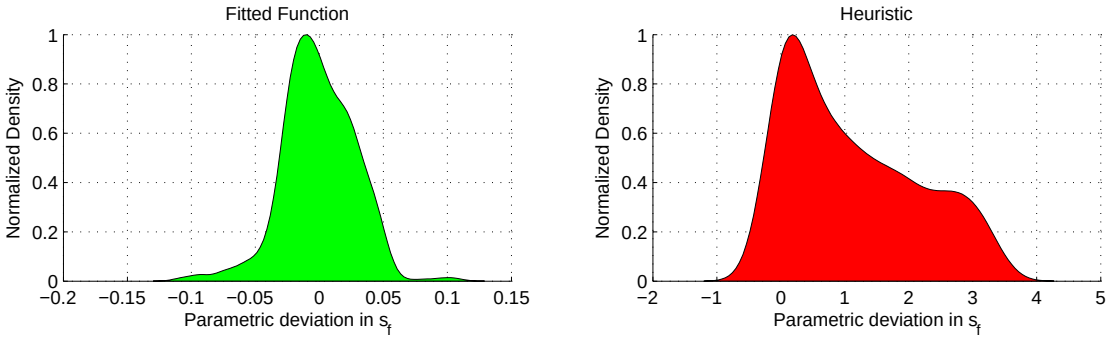
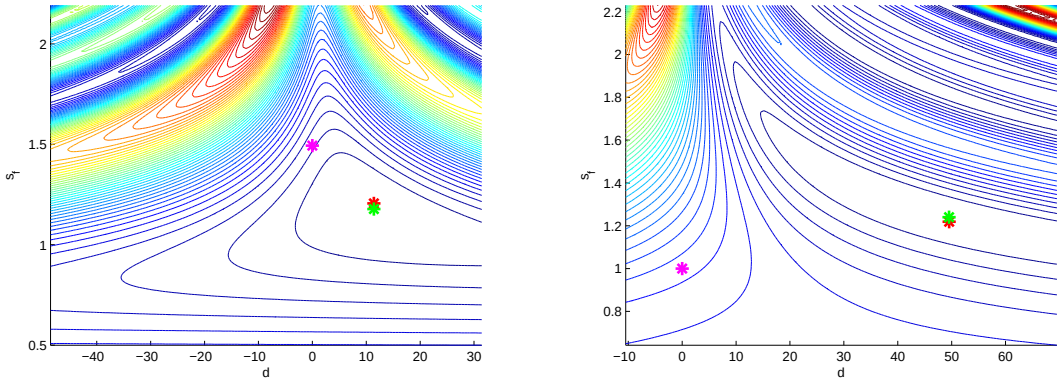


Figure 4.30: Parametric distances between initial guess and minima in the s_f dimension

in Figure 4.31



(a) $\varphi_1 = 90°$ (b) $\varphi_1 = 45°$

Figure 4.31: Example of quality differences between initial guesses. The green star shows the location of the desired minima, the red star initial point retrieved from the neural network and the purple that given from the proposed heuristic. The problem shown is the basic constraint satisfaction case (i.e. no optimality measure introduced) $x_1 = 10$, $y_1 = 15$, $\kappa_1 = \kappa_0 = 0$. The terminal heading φ_1 is set to 90 and 0 for (a) and (b) respectively

4.2.2 Using Curvature as a Cost Function

To demonstrate the effects of introducing a measure of optimality to the problem, the solution is sought after using the cost function from Equation 3.45. The problem is studied for three different settings of free parameters, all scaled following the method in D.3 and the effects both on computational performance and trajectory appearance is sought. For numerical stability reasons the lightweight solver written specifically for the constraint satisfaction case have been avoided. Instead the native MATLAB implementation of the

Levenberg-Marwuardt algorithm through the function *fsolve* is utilized. All numerical integrations are performed with the adaptive step size version of *Simpsons Composite Quadrature Rule* using the built in function *quad*. With this, easier convergence and more accurate results are achieved. The algorithm starting point, i.e. the initial guess, is taken as the result from the basic case and the introduced free parameters are set to zero. With this, the Lagrange multipliers are initiated using Equation 3.44. Given that the formulation is based solely on first order optimality conditions, no assertions on the characteristics of the extreme point found can be made due to which the development of the cost is examined. For the one, two and three degrees of freedom (DOF) cases, this is shown in Figure 4.32. Note especially how the one DOF case converges to a local maximum in the cost. Even though the number of needed iterations varies between solutions for different boundary conditions the discrepancy towards the basic case is in general large, c.f. Figure 4.34(b). Furthermore, each introduced DOF generally induces a larger number of iterations needed to converge, and thereby consumed time.

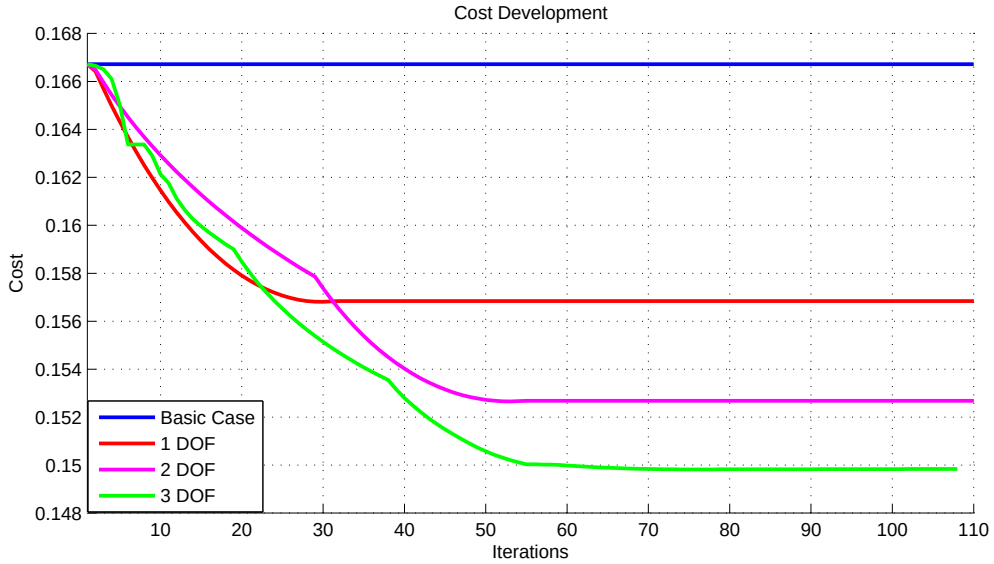


Figure 4.32: Development of the cost $J = \int_0^{s_f} \kappa^2(s)ds$ for solutions in one, two and three degrees of freedom for a simple left turn.

The corresponding trajectories are shown in Figure 4.33 together with their corresponding curvature functions $\kappa(s)$. It should be noted here that the boundary conditions for which the results are obtained, $x_1 = 15, x_2 = 10, \kappa_1 = 0, \kappa_0 = 0$ and $\varphi_1 = 90°$ belongs to a subset of the envelope presented in 4.2.1 that actually yielded convergence and that a failure to converge was observed for the more extreme cases . The general robust performance seen in the basic case was not present when the free parameters were introduced. Further, the similarity between the optimized trajectories seen in Figure 4.33 are a good representation of the results found. With each degree of freedom n introduced,

the difference to the result from using $n-1$ decreases.

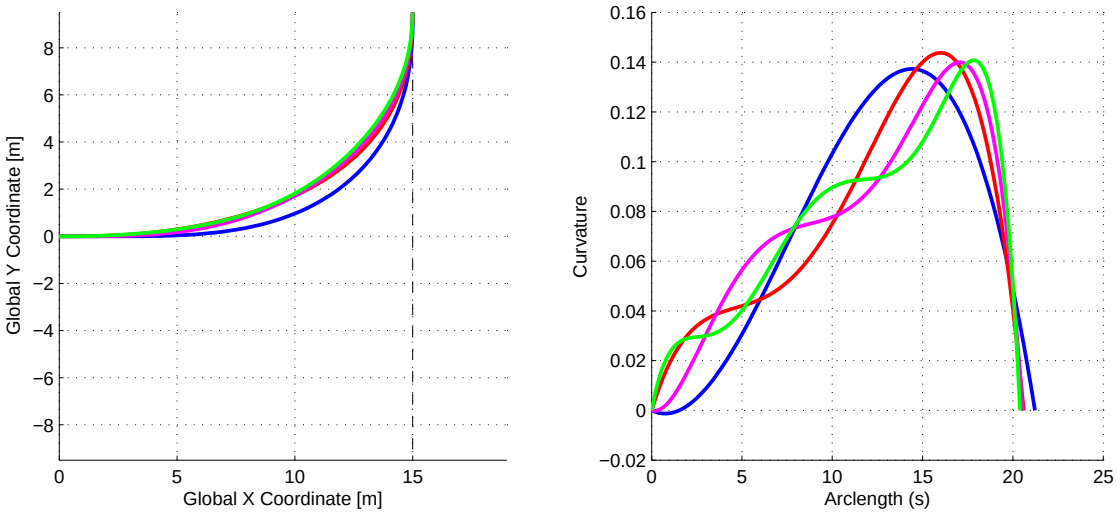
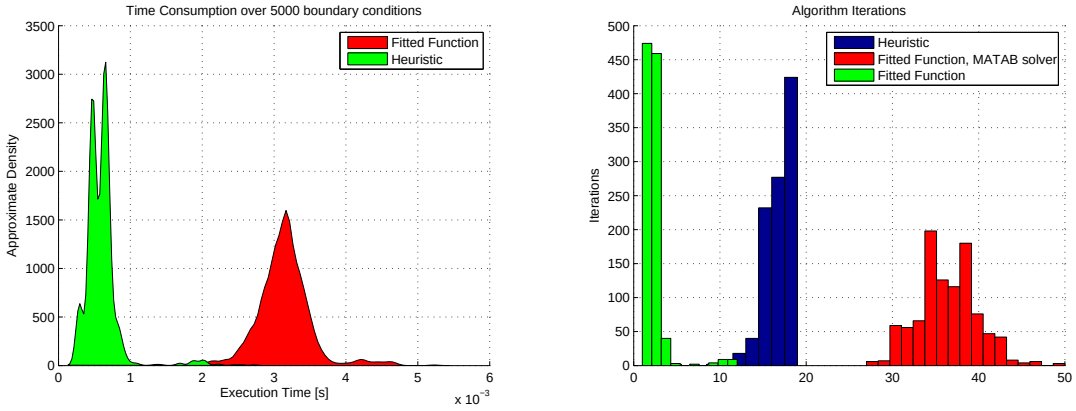


Figure 4.33: Example of resulting trajectory when the measure presented in Equation 3.45 is used. Following the color coding presented in Figure 4.32, the basic case is blue and the red, magenta and green ones represent the one, two and three degrees of freedom trajectories respectively.

4.2.3 Performance

To examine the time performance of the basic trajectory generator repeated runs over the described envelope is again performed. The intention is to determine the gains in using a lightweight implementation of the solution algorithm and an accurate initial guess. The effects of solver choice and initial guess generation are isolated by solving each boundary condition using: 1) Both accurate initial guess and lightweight solver, 2) Lightweight solver and heuristic based initial guess and 3) Robust solver with adaptive integration and accurate initial guess, and compare the results. The ”robust optimization” is here taken as the Matlab implementation of the Levenberg-Marquardt algorithm through the function *fsolve* coupled with the *Adaptive Simpson Quadrature* in the function *quad*. The results in terms of time consumption and needed iterations are summarized in Figure 4.34 where 5000 boundary conditions randomly selected from the presented envelope have been used. Note that the time results for the robust solver has been removed in 4.34(a) for visualization reasons. The average time required for this combination lies around 0.3 s and would necessitate a large plot to visualize the difference between the other two cases. An additional remark regarding the Matlab implementation is that it only uses the actual function, i.e Equation 3.25, and no explicit statements on the jacobian and gradients. These are instead approximated using finite differences resulting in a much higher number of function evaluations and thereby longer execution times.



(a) (b)

Figure 4.34: Time consumption and needed number of iterations for 5000 randomly chosen boundary conditions within the envelope described.

5

Discussion

This chapter contains discussions and analysis of the results presented above. For the highway planner, the scenarios and trajectory selection are treated in the general while the findings regarding computational demands and model fidelity are examined in detail. For practical reasons discussions around the four specific scenarios are placed in conjunction with their respective plots in Chapter 4. Further, some issues and problems as well as the means by which they can be mitigated are also included as well as some notes on parameter settings. The section on turning are analysed in terms of its performance and computational benefit, where specific treatment is given the introduction of cost functional and how these affect the resource requirements.

5.1 Selection system and trajectories for Highway

The general results from 4.1 implies that the proposed system indeed has the potential to handle relatively complex scenarios on structured roads in a safe manner. At its core, the selection process introduced in 3.4 works rather good, and shows a consistent behaviour when exposed to various environment challenges in simulation. The cost development in all scenarios shows that the use of differentially formed, or filtered, linear combinations of costs a plausible tool for trajectory selection in path planning.

Discontinuities resulting from trajectory switching Some issues exist however, specifically for more complex scenarios where the trajectory selection for some reason is forced to abort a commenced manoeuvre in favour of another. Discontinuities are as previously noted then introduced in the lateral jerk with large effects on the smoothness

of the corresponding acceleration. At this point it is not clear how these effects will be perceived by a person in the vehicle for which qualitative experimental results are required.

The difficulties to extrapolate the results to an actual implementation is increased further since the characteristics of the assumed path following controller executing the trajectory and its interaction with the vehicle dynamics is unknown. If ideal tracking is assumed, the non linear bicycle model as stated in Equation E.4 indicates relation between lateral jerk and the steering angle rate $\dot{\delta}$, angular- and lateral accelerations $\dot{\omega}_z, \dot{v}_x$. Following the results from Appendix B.1, the two latter quantities are likely to influence the perceive ride quality adversely. Moreover, more or less discontinuous changes in $\dot{\delta}$ are likely to be perceived as strange and therefore risk damaging the passenger trust in the system. It should of course be noted that all physical system more or less exhibit low pass behaviour, and that discontinuities in the strict sense are impossible. The proposed system easily admits a minor modification that mitigates this issue. As the trajectory generated in essence is a polynomial in time, the order to which it is endpoint continuous is determined solely by its order. One could therefore augment Equation 3.3 to achieve essentially continuity in arbitrarily high derivatives of position. By this, a jerk continuous trajectory is simply a sextic (6th order) polynomial, with coefficients determined by the boundary conditions. However, it should be noted here that the jerk minimizing property derived in 3.2 no longer holds if this solution is utilized. The sextic polynomial can instead be shown to minimize *snap*, or fourth derivative of position. Unfortunately, this can potentially worsen the experienced ride quality, as is discussed in Appendix B.1. A second alternative is to reformulate the problem as the minimisation of the integral of squared jerk subjected to endpoint constraints of sufficiently high order. This option has not been studied in detail, but although its solution lacks the simplicity of the polynomial, it has an analytical form and could therefore be integrated into the planning framework.

Getting stuck at "local extrema" A second issue relating to the selection system is the choice of comparison dimensions and their respective parameters. The different cost measures used might be insufficient to guarantee adequate performance in all situations. The set-up used with parameters according to Table 4.2, had for some more complex scenarios than those presented problems when it after a lane change ended up too close behind a preceding vehicle. Since high proximity values constantly were punished hard and the ACC model managed the velocity, the host vehicle effectively got stuck behind the obscuring vehicle, lacking manoeuvring capabilities. In effect, this corresponds to reaching a local extrema of the overall objective. Similarly, the punishment against rapid switching sometimes forced the vehicle to remain longer than necessary at non center latitudes. However, the selection process is easily augmented or modified to account for other measures of quality and a careful designer could with ease tune the system to desired performance characteristics.

Time Performance With the Matlab code diagnostics tool it is as expected seen that roughly 70% of the execution time is spent in the simulation of longitudinal trajectories, i.e prediction of the environment and corresponding ACC model reaction. Large benefits could therefore be made in leaner implementation of this particular component. Examining the development of the mean execution in both cases implies a linear average time complexity when coupling effects are neglected, c.f. Figure 4.21. Even as the latter is included, almost linear behaviour is observed. Though linear, the increment of trajectories is considerably more expensive in terms of computation than that of obstacles. Fortunately, the need for more than three trajectories per lane was found unnecessary, whereby the largest cost can be avoided. Further, in the presented simulations all defined obstacles, both in and out of sensor range, are checked in the prediction step. As a result, parts of the consumed time lies in the treatment of obstacles that does not affect the behaviour of the simulated vehicle. Consequently, a leaner implementation where only "seen" obstacles are treated would lower the rate of change in average time consumption. The increased variance observed in Figure 4.19 is likely related to the modelling of the ACC target selection, where the frequent switching induced by the particular scenario injects disparity. Combined, these properties should indicate good scalability possibilities, specifically if the system is designed to use a low number of trajectories. It should further be noted that the program written is relatively large and Matlab for these purposes is a rather clumsy tool, an actual implementation where care has been taken performance would probably be orders of magnitude faster. In spite of this, the performance is by all means good and could very likely be used on target hardware.

Model Validation The results of the extensive DYMOLA simulations from 4.1.3 shows a clear correlation between the "aggressiveness" of a motion and the error relative the simple model, where the extreme cases in the *high velocity-low distance* region of the error renders the absolute highest. For all intents, the manoeuvres chosen should therefore be as nice as possible if the particle model predictions are to be trusted. When examining the state evolution over the extreme cases it is found that the vehicle essentially broadsides through the trajectory in manner incompatible with regular driving further strengthening this.

Further, the comparison seems to be valid over the normal manoeuvring range when considering the slip angle limit. Additional limits are easily introduced, but lie outside the scope of this thesis and will therefore not be discussed. It should be noted that a designer easily can specify a limit on one of the quality measures, whereby the acceptable region is defined by the area restricted by its corresponding isoline and the slip angle limit. However, as a thorough investigation on what quality level the planner needs constitutes a large body of work in itself, discussions on the precise numerical values are avoided. However, the sign of the error in the measure E_{max} is important for the proposed planner. As implied by Equation 4.8 $E_{max} < 0$ corresponds to underestimates of the actual acceleration. Interestingly, not one of the 4300 cases tested results gave this result. Even though no formal guarantees can be stated due to the empirical nature of

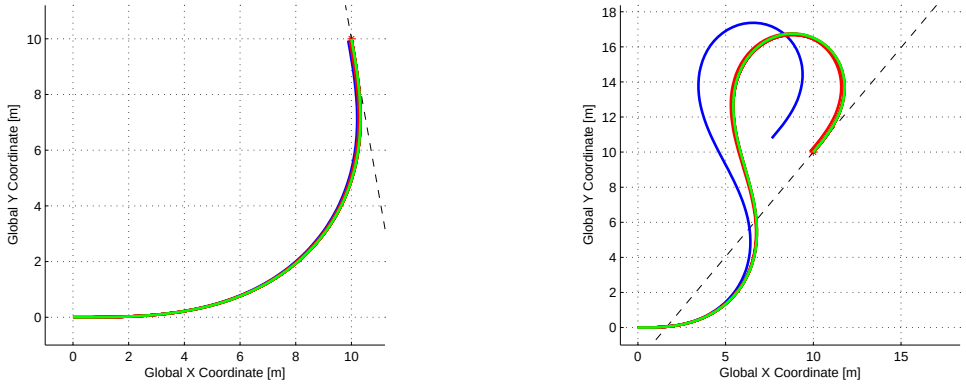
the evaluation, the result together with the apparent smoothness of E_{max} indicates that $a^A \leq a^S$ for all possible parameter combinations. By this, the limitations imposed on the simple model by Algorithm 5 in Appendix refap:highway can also be said to apply, albeit conservatively, to the actual vehicle.

5.2 Trajectories for Turning

Opposed to the more general planning system discussed in the previous section, the trajectory generator for the three dimensional (x,y,φ) setup presented in this thesis has merely been examined on its execution characteristics and ability to generate accurate trajectories. It is however emphasized that there in principle is no problem to integrate the described trajectories in to the selection system. The repeated simulations performed indicates that given a sufficiently well fitted function as provider of initial guesses, the algorithm safely delivers a correct (convergent to the right minima) result over the entire relevant parameter envelope. Further, the benefits of the scaling procedure mentioned in D.3 are also evident as no numerical issues arise. In the general situation, i.e. during normal turns, the applied polynomial curvature expression is smooth enough to render extremely cheap calculations of the integrals involved, needing no more than roughly 10 Simpson Quadrature intervals a piece.

Initial Guess Generation The initial guess is essentially a good solution in itself, and in most regions of the tested boundary condition envelope the numerical procedure simply refines its appearance, making very small adjustments. Similarly, the dimensional reduction achieved by reformulating the problem to the two s_f,d through inclusion of boundary conditions on curvature and direction forces the last two to always be satisfied. That is, regardless of other parameters, the endpoint curvatures κ_1,κ_0 and headings $\varphi_0 = 0, \varphi_1$ will always be correct as visualized in Figure 5.1. Combined, these two steps gives that the algorithm simply moves the endpoint of an already correctly directed and nearly correctly shaped trajectory towards the goal until the set tolerance is reached. The process is exemplified in Figure 5.1 where the error is set in the scaled coordinates to $\varepsilon_S = 10^{-8}$. The problem scale d is roughly 14 m due to which the actual error measure is $\varepsilon = \varepsilon_S d \approx 1.4 \cdot 10^{-7} m$. It should be noted that this extreme sub micrometer precision is merely for visualization purposes since the algorithm mostly completes in very few iterations, hardly suitable for illustration.

For some inputs to the neural network function, i.e. boundary conditions of Equation 3.29, notably those where $\kappa_0 \approx 0$ and the end position and heading are set such that no large bends are forced on the trajectory, the initial guess is good enough to achieve a reasonable tolerance directly or within one iteration. As these manoeuvres comprises most normal situations the generation algorithm is essentially instant for normal cases. This is also shown in the time results of the repeated simulations, where the the neu-



(a) (b)

Figure 5.1: Visualization of algorithm convergence. Initial guesses as given from the fitted neural network are coloured blue, steps in iteration red and optimal trajectory green. In both plots $\kappa_1 = \kappa_0 = 0$, while (a) has $x_1 = y_1 = 10$ with $\varphi_1 = 100°$ and (b) $x_1 = y_1 = 10$ with $\varphi_1 = -130°$

ral network function clearly outperforms the simple heuristic proposed in the literature. The nature of using an interpreted language and the general Matlab overhead significantly pushes up the average time spent, ending up around 0.5 millisecond. However, implemented directly in C with explicit memory handling and code optimization this method would essentially be free computationally. As a sidenote, it is therefore evident that the performance of the trajectory generator present herein vastly outperforms the equivalent presented in [3] on which it is based. The problem formulation presented in this paper, i.e. Equation 3.24, has been used by a number o authors, e.g. [12] and [5], as the trajectory generation choice in state lattice planners, and it is evident that the gains of using a accurate initial guess go beyond the simple turning manoeuvres presented here. Especially since the graph building, or trajectory generating phase in these planners commonly is the most time consuming component and generally involves hundreds or thousands of function calls.

Use of cost a Functional The use of cost functional present an attractive way of online tuning the trajectory preference. However, the results where these are introduced are unfortunately not as attractive. In essence similar to the basic case, the non-linear problem is troubled with divergence, frequent convergence to wrong minima or even local maxima, or numerical issues (e.g. ”flat” residual regions) where search directions become small. Even when correct convergence occurs, it usually requires an unacceptable amount of iterations and its consumed time, being in the order of seconds, makes it far from suitable for real time implementation. As mentioned in the previous section, the initial guesses are however constructed naively by setting the additionally introduced

free parameters to zero. When examining the results of successful runs, one finds that this commonly is far from the desired value. In principle it should therefore be possible to utilize the same neural network based procedure as for the basic case and thereby retrieve significantly better results. In principle, the training set could be generated with a global optimization algorithm, e.g. a Genetic or Swarm based one, since the optimality criterion introduces the presence of a *global* minima. Although the high dependence on good initial guesses by far is the most important adverse aspect, another exists in the increased usage of numerical integration and algebraic operations which essentially grows with the cube of the free parameters. As the positive effects, or impacts on the resulting trajectory, also seem to lessen with each introduced degree of freedom, c.f. Figure 4.32, it seems unwise to use more than two free parameters.

As a note on the representative power of the assumed polynomial form of the control function, it is far from evident that it is sufficient when optimality is introduced. Examination of Figure 4.33, specifically the appearance of the curvature functions indicates that the polynomial representation might be to stiff and thereby yield unnatural behaviour. Considering curvature proportional to lateral acceleration and thereby steering angle, it is evident that the trajectory is far from what would have been taken by a skilled human driver.

6

Conclusions

In the presented work, the problem of path planning was studied and adaptation of state of the art approaches to automotive applications were investigated. Further analysis revealed that, given the constraint set for this particular work, commonly used path planning techniques cannot be directly applied. Hence, design procedure had to partially stray from the norms. Following a modular approach, separate solutions tailored for highway driving and turning were developed. Performance and satisfaction of the demanded criteria was confirmed via simulations.

As it was shown, decoupling decision making, path planning and control in this context leads to unnecessary limitations. Requiring an external controller, in the form of ACC for regulating longitudinal velocity, as well as presence of a decision making module in the described form, reduces the path planning problem into that of short-term trajectory generation. Of course, in this specified framework and for the intended applications in this work, such a design promises to be effective.

Simulation results detailed in Chapter 4 demonstrated the viability of the proposed solution for the intended purposes. Mainly, analysis results of the time complexity affirms that this solutions addresses the concerns on computational resources. Moreover, the motion models used, especially for the highway manoeuvres, prove to be credible for the range of intended operations. This satisfies the important criterion of kinematic feasibility.

As discussed in Section 2.4, the modular approach adopted in this work is known to favourably fit into the development process in practice for automotive functions of comparable nature. Such functionalities need to pass through extensive stages of experimental validation to be deemed suitable for production. A modular design facilitates ease of

these verification steps and in turn a more agile development loop.

However, being a relatively add-hoc design, the provided solution has limited flexibility and scalability to a larger range of use-cases. for a more versatile path planner acting on longer planning horizons, the constraints present in this framework need to be softened. Mainly, allotting more computational resources to a planner with more authority in the form of e.g. decision making and velocity profile generation will lead to a more versatile solution able to handle a variety of future functionalities, e.g. driving in unstructured environments and low-speed settings with complicated manoeuvres such as parking lots. This approach will presumably be more cost-effective in long term.

Future Work

As a continuation of the work presented here in, experimental validation is proposed for further examination of model fidelity, applying both to the particle model used for highways and the soap box representation in turning. The problems highlighted in Moreover, efforts should be made to ensure a smooth switching between the separate modes of operation, which have been left out of the work presented herein. Additionally, the closed loop performance of the highway system needs to be further evaluated, specifically in the presence of measurement noise and non ideal tracking. The benefits of increased model detail in the prediction step should also be studied with inclusion of lateral obstacles movements and models of driver behaviour. Concerning the manoeuvres for turning, a similar DAE validation is recommended as a first step, followed by actual vehicle testing. Further, the possibility to apply the same input parametrization and solution procedure to a more accurate model, e.g. a linear bicycle representation, and its effect on the representative power of the generated trajectories examined. If possible, the results could potentially enhance the usefulness of the method. In the general context of autonomous drive, large efforts need to be put into examining different measures of comfort and how various phenomena is perceived by the passengers. The lack of research done on this topic is surprising, and the field seems largely to be open. Specifically, numerical statistics should be of interest on which the enforced bounds could be based. Possible other results from such research could be updated or augmented optimization criteria for generation of the actual trajectories as well as enhancements to other highly influential components. In the context of this thesis, tuning of the proximity measure or selection method introduced in 3.4 exemplifies the latter.

Bibliography

[1] J. Barraquand, J.-C. Latombe, Nonholonomic multibody mobile robots: controllability and motion planning in the presence of obstacles, in: Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on, 1991, pp. 2328–2335 vol.3.

[2] S. M. LaValle, Planning Algorithms, Cambridge University Press, Cambridge, U.K., 2006, available at http://planning.cs.uiuc.edu/.

[3] A. Kelly, B. Nagy, Reactive nonholonomic trajectory generation via parametric optimal control, The International Journal of Robotics Research 22 (7 - 8) (2003) 583 – 601.

[4] T. M. Howard, C. J. Green, A. Kelly, State space sampling of feasible motions for high performance mobile robot navigation in highly constrained environments (2007).

[5] M. Pivtoraiko, A. Kelly, Differentially constrained mobile robot motion planning in state lattices (2008).

[6] S. Karaman, E. Frazzoli, Sampling-based algorithms for optimal motion planning, Int. J. Rob. Res. 30 (7) (2011) 846–894.

[7] A. Takahashi, T. Hongo, Y. Ninomiya, G. Sugimoto, Local path planning and motion control for agv in positioning, in: Intelligent Robots and Systems '89. The Autonomous Mobile Robots and Its Applications. IROS '89. Proceedings., IEEE/RSJ International Workshop on, 1989, pp. 392–397.

[8] T. Hiraoka, T. Kunimatsu, . Nishihara, H. Kumamoto, Modeling of driver following behavor ubased on minimum-jerk theory, 12th World Congress on ITS, November 2005, San Fransisco 1 (1) (2005) 4–27.

[9] B. Jacobson, Vehicle Dynamics, Compendium for Chalmers Course MMF062, 2012.

[10] S. Kammel, J. Ziegler, B. Pitzer, M. Werling, T. Gindele, D. Jagzent, J. Schröder, M. Thuy, M. Goebl, F. v. Hundelshausen, O. Pink, C. Frese, C. Stiller, Team annieway's autonomous system for the 2007 darpa urban challenge, J. Field Robot. 25 (9) (2008) 615–639.

[11] A. Bacha, C. Bauman, R. Faruque, M. Fleming, C. Terwelp, C. Reinholtz, D. Hong, A. Wicks, T. Alberi, D. Anderson, S. Cacciola, P. Currier, A. Dalton, J. Farmer, J. Hurdus, S. Kimmel, P. King, A. Taylor, D. V. Covern, M. Webster, Odin: Team victortango's entry in the darpa urban challenge, J. Field Robot. 25 (8) (2008) 467–492.

[12] C. U. et. al., Autonomous driving in urban environments: Boss and the urban challenge, Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I 25 (8) (2008) 425–466.

[13] D. Dolgov, S. Thrun, M. Montemerlo, J. Diebel, Practical search techniques in path planning for autonomous driving, in: Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08), AAAI, Chicago, USA, 2008.

[14] M. Wang, T. Ganjineh, R. Rojas, Action annotated trajectory generation for autonomous maneuvers on structured road networks, in: Automation, Robotics and Applications (ICARA), 2011 5th International Conference on, 2011, pp. 67–72.

[15] P. Resende, F. Nashashibi, Real-time Dynamic Trajectory Planning for Highly Automated Driving in Highways, 13th International IEEE Annual Conference On Intelligent Transport Systems,Madeira, Portugal, 2010.

[16] S. Glaser, B. Vanholme, S. Mammar, D. Gruyer, L. Nouveliere, Maneuver-based trajectory planning for highly autonomous vehicles on real road with traffic and driver interaction, Intelligent Transportation Systems, IEEE Transactions on 11 (3) (2010) 589–606.

[17] M. Werling, J. Ziegler, S. Kammel, S. Thrun, Optimal trajectory generation for dynamic street scenarios in a frenet frame, in: Robotics and Automation (ICRA), 2010 IEEE International Conference on, 2010, pp. 987–993.

[18] D. Kogan, R. Murray, Optimization-based navigation for the darpa grand challenge, 2006 Conference on Decision and Control.

[19] I. Papadimitriou, M. Tomizuka, Fast lane changing computations using polynomials, in: American Control Conference, 2003. Proceedings of the 2003, Vol. 1, 2003, pp. 48–53 vol.1.

[20] M. T. Wolf, J. Burdick, Artificial potential functions for highway driving with collision avoidance, in: Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on, 2008, pp. 3731–3736.

[21] M. McNaughton, Phd thesis: Parallel algorithms for real-time motion planning (July 2011).

[22] N. Andréasson, A. Evggrafov, M. Patriksson, An introduction to continuous optimization, Lightning Source Incorporated, 2005.

[23] S. Haykin, Neural Networks and Learning Machines (3rd Edition), Prentice Hall, 2008.

[24] M. Wahde, A general-purpose method for decision-making in autonomous robots., in: B.-C. Chien, T.-P. Hong, S.-M. Chen, M. Ali (Eds.), IEA/AIE, Vol. 5579 of Lecture Notes in Computer Science, Springer, 2009, pp. 1–10.

[25] H. B. Pacejka, Tire and Vehicle Dynamics, Society of Automotive Engineers, Incorporated, 2006.

[26] L. E. Dubins, On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents, American Journal of Mathematics 79 (3) (1957) pp. 497–516.

[27] J. A. Reeds, L. A. Shepp, Optimal paths for a car that goes both forwards and backwards., Pacific Journal of Mathematics 145 (2) (1990) 367–393.

[28] J. Schwartz, M. Milam, On-line path planning for an autonomous vehicle in an obstacle filled environment, in: Decision and Control, 2008. CDC 2008. 47th IEEE Conference on, 2008, pp. 2806–2813.

[29] P. Hart, N. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, Systems Science and Cybernetics, IEEE Transactions on 4 (2) (1968) 100–107.

[30] N. Nilsson, Principles of Artificial Intelligence, Symbolic Computation / Artificial Intelligence, Springer, 1982.

[31] E. W. Dijkstra, A note on two problems in connexion with graphs., Numerische Mathematik 1 (1959) 269–271.

[32] J. Ziegler, M. Werling, J. Schroder, Navigating car-like robots in unstructured environments using an obstacle sensitive cost function, in: Intelligent Vehicles Symposium, 2008 IEEE, 2008, pp. 787–791.

[33] P. Svestka, M. H. Overmars, M. H. Overmars, M. H. Overmars, Motion planning for car-like robots using a probabilistic learning approach, Int. J. of Robotics Research 16.

[34] A. Stentz, I. C. Mellon, Optimal and efficient path planning for unknown and dynamic environments, International Journal of Robotics and Automation 10 (1993) 89–100.

[35] A. Stentz, The focussed d* algorithm for real-time replanning, in: In Proceedings of the International Joint Conference on Artificial Intelligence, 1995, pp. 1652–1659.

[36] S. Koenig, M. Likhachev, Improved fast replanning for robot navigation in unknown terrain, in: in Proceedings of the International Conference on Robotics and Automation, 2002, pp. 968–975.

[37] D. Ferguson, M. Likhachev, A. Stentz, A guide to heuristicbased path planning, in: in: Proceedings of the Workshop on Planning under Uncertainty for Autonomous Systems at The International Conference on Automated Planning and Scheduling (ICAPS, 2005.

[38] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, S. Thrun, Anytime dynamic a*: An anytime, replanning algorithm, in: In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS, 2005.

[39] L. Kavraki, P. Svestka, J.-C. Latombe, M. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, Robotics and Automation, IEEE Transactions on 12 (4) (1996) 566–580.

[40] S. LaValle, J. Kuffner, J.J., Randomized kinodynamic planning, in: Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on, Vol. 1, 1999, pp. 473–479 vol.1.

[41] P. Blaer, Robot path planning using generalized voronoi diagrams (May 2013).
URL `http://www.cs.columbia.edu/~pblaer/projects/path_planner/`

[42] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots, in: Robotics and Automation. Proceedings. 1985 IEEE International Conference on, Vol. 2, 1985, pp. 500–505.

[43] T. Hellström, Robot navigation with potential fields (2011).

[44] M. Pivtoraiko, A. Kelly, Efficient constrained path planning via search in state lattices (2005).

[45] T. Flash, N. Hogan, The coordination of arm movements: An experimentally confirmed mathematical model, Journal of Neuroscience 5 (1985) 1688–1703.

[46] N. Hogan, Adaptive control of mechanical impedance by coactivation of antagonist muscles, Automatic Control, IEEE Transactions on 29 (8) (1984) 681–690.

[47] M. J. E. Richardson, T. Flash, Comparing smooth arm movements with the two-thirds power law and the related segmented-control hypothesis, Journal of Neuroscience 22.

[48] R. Shadmehr, S. P. Wise, A mathematical muscle model in supplementary documents for computational neurobiology of reaching and pointing (2005).

[49] O. Rohrer, S. Fasoli, H. I. Krebs, R. Hughes, B. Volpe, W. R. Frontera, J. Stein, N. Hogan, Movement smoothness changes during stroke recovery, Journal of Neuroscience 22 (2002) 8297–8304.

[50] C. Fernandez, J. Goldberg, Physiology of peripheral neurons innervating otolith organs of squirrel-monkey, Journal of Neurophysiology 39 (1976) 996–1008.

[51] S. J. R. Benson A. J., Spencer M. B., Thresholds for the detection of the direction of whole-body, linear movement in the horizontal plane, Aviation, Space, and Environmental Medicine 57 (1986) 1088–1096.

[52] P. R. Grant, B. Haycock, Effect of jerk and acceleration on the perception of motion strength, Journal of Aircraft 45 (2008) 1190–1197.

[53] N. A., G. P. R., D. P., Modeling the perception of acceleration and jerk using signal detection theory, in: AIAA Modeling and Simulation Technologies Conference and Exhibition, 2008.

[54] J.-J. Martinez, C. Canudas-de Wit, Model reference control approach for safe longitudinal control, in: American Control Conference, 2004. Proceedings of the 2004, Vol. 3, 2004, pp. 2757–2762 vol.3.

[55] L. Hoberock, U. S. D. of Transportation. Office of University Research, A Survey of Longitudinal Acceleration Comfort Studies in Ground Transportation Vehicles, Research report, Council for advanced transportation Studies, University of Texas at Austin, 1976.

[56] N. Hoboken, Urban Transit Systems and Technology, John Wiley and Sons, 2007.

[57] D. Madås, M. Nosratinia, M. Keshavarz, P. Sundström, R. Philippsen, A. Edehall, K. Dahlen, On path planning methods for automotive collision avoidance (2012).

[58] R. Adams, Calculus, Pearson plc, 2006.

A

Method Survey

This chapter contains a survey of the most important approaches taken in both industrial and academic attempts to attack the automotive path planning problem. The different classes of algorithms are described together with their gains ans shortcomings as noted in the literature. Important aspects such as computational load, ease of tuning and implementation, the ability to explicitly include feasibility guarantees and limits on the resulting trajectory and algorithmic completeness are treated.

A.1 Optimal Control

The general form of the path planning problem, formulated as an optimal control problem as presented in 2.2.2 has been the topic of extensive research. This problem was initially investigated in 1957 in the work of Dubins [26] where it was shown that in an obstacle-free, two dimensional euclidean space, an optimal minimum-length path with constraints on curvature and with boundary conditions can be realized by a series of *bang-bang* controls between a limited selection of forward motions, namely straight, full turn to the right and full turn to the left. The model with such motion capabilities is referred to as *Dubins' car*. This observation was later extended to *Reed-Shepp's car* [27] which is also capable of reversed motion.

In the presence of changing obstacles and considering non-linear dynamics, the above-mentioned optimal control problem proves to be challenging. Current solutions which approximate the original formulation are computationally demanding even on today's processors. As an example of a practical implementation, the work in [28] can be considered. Their solution generates optimal trajectories both in presence of linear and

non-linear constraints in form of boundary polynomials and obstacle ellipsoid, respectively. Moreover, differential constraints on vehicle steering angle and steering angle rate are incorporated. In their solution, samples of the *flat* variables and their derivatives are used as independent variables in a non-linear programming algorithm.

A.2 Search Based Discrete Methods

Application of graph search methods in the area of path planning for mobile robots has been extensively studied. The basic thought behind this approach is to represent the search space by a graph where the edges are assigned a cost for travelling between two vertices. Each vertex corresponds to either a location in the spatial plain or a state of the discretized state-space. The former, however, typically proves to be insufficient since the nonholonomicity and the dynamics of the mobile robot are overlooked and therefore can result in an untraversable path. Discretizing an extended state-space of the robot whilst accounting for the differential constraints provide kinematic feasibility of the path while sacrificing the execution speed of the graph search methods. What follows introduces some of the most common graph search methods used in path planning for mobile robots.

A.2.1 A* Algorithm

The A* search method [29][30] was first introduced in 1968 as a modified version of the classic Dijkstra's algorithm [31]. A* improves Dijkstra's algorithm by including a heuristic which guides the search direction and by doing so decreases the number of necessary computations.

The cost function for this search method can be described as

$$f(n) = g(n) + h(n) \tag{A.1}$$

where $g(n)$ is the sum of the costs from the initial node up to the current node and the heuristic function $h(n)$ represents the future cost from the current node to the goal. $h(n)$ never overestimates this cost and in other words is an *admissible* heuristic. For instance, assuming a search in the 2-dimensional spatial plain, let $[x_{goal}\ y_{goal}]$ be the position of the goal state n_{goal}. Now, for a node n with coordinates at $[x\ y]$, $h(n)$ can simply be defined as $\sqrt{(x_{goal}-x)^2+(y_{goal}-y)^2}$ which disregards the obstacles and hence is an underestimate of the actual cost, distance to n_{goal} in presence of obstacles. With such underestimating heuristic, A* guarantees optimality, i.e. finding the path with the least cost-to-go. Choosing a suitable heuristic is of significant importance in this graph search method. A large difference between the underestimated heuristic cost and the actual cost can degenerate the time complexity of A* to that of an exponential time algorithm [32].

Algorithm 2 demonstrates A* in further detail. The algorithm receives nodes $n \in N$ where N is a set of all the possible robot positions and returns the path $TraversedPath$ from the initial position n_{start} to the designated goal n_{goal}. Function EdgeCost(n_i,n_j) represents the cost of a single edge between the two adjacent nodes n_i n_j. $h[n_i]$ is the above-mentioned heuristic cost from n_i to the goal position n_{goal}. $g[n_i]$ and $f[n_i]$ represent the costs at the node n_i as described above. The algorithm starts by assigning an infinite cost g to all the nodes except for the initial cost $g\,[n_{start}]$, which holds a value of 0. n_start is then copied to a *priority queue* $OpenSet \subset N$ which is an ordered set where the element n with the highest priority is the one with the assigned cost $f[n]$. As the algorithm progresses, the highest priority element in $OpenSet$ is removed from the set and the costs for the neighbouring nodes $n_{neighbour}$ are evaluated. If the sum of the current cost at node $n_{current}$ and EdgeCost($n_{currrent}$,$n_{neighbour}$) is less than the current cost of a neighbouring node, the corresponding cost for that node $f[n_{neighbour}]$ is set to this new value, and the affected node $n_{neighbour}$ is placed in $OpenSet$. This process repeats until n_{goal} is reached or $OpenSet$ is empty, which means that the goal cannot be reached. Note that the returned value $TraversedPath\,[n_{current}]$ is in fact a set of nodes building up the path from n_{start} to n_goal. This implies that at every iteration, the predecessor of each node should be saved in order to be able to recover the final path. For the sake of simplicity, this is not explicitly handled in Algorithm 2.

The *dynamic programming* nature of the A* method promises efficient implementation of the algorithm and hence a relatively fast execution. However, in the context of path planning for mobile robots and in the case of employing A* only in the spatial domain, disregarding the nonholonomic constraints renders the resulting crude path far from desirable. Therefore, in practice the resulting path is typically used as a heuristic in more sophisticated hybrid methods or is fed to some smoothing stage to guarantee satisfying the differential constraints.

Some practical examples of utilization of A* in path planning for mobile robots can be seen in [13] and [32]. In [13], an A* variant, the so-called *Hybrid-State* A* is applied to the discretized 3-dimensional kinematic state-space of the vehicle. The search is guided by a combination of heuristics to take into account both the obstacles in the search space and the nonholonomic constraints of the vehicle. The resulting path which is guaranteed to be drivable is then used in a gradient decent algorithm to locally improve the trajectory. Similarly, [32] employs A* on a descretized state-space of the vehicle while a heuristic based on rotation-translation-rotaion (RTR) [33] accounts for kinematic constraints and another heuristic built upon a *Voronoi* diagram (see A.4.1) incorporates the obstacles in the search.

A.2.2 D* Algorithm

In a real-world scenario, a mobile robot typically has limited knowledge about the obstacles on the path leading to the goal. This knowledge is incrementally improved as the

Data: all nodes $n \in N$, start node n_{start}, goal node n_{goal}
Result: $TraversedPath$ path from n_{start} to n_{goal}
$OpenSet \leftarrow n_{start}$;
$TraversedPath \leftarrow \emptyset$;
forall the $n \in N$ **do**
  $g[n] \leftarrow \infty$
end
$g[n_{start}] \leftarrow 0$;
$f[n_{start}] \leftarrow g[n_{start}] + h(n_{start})$;
$OpenSet \leftarrow n_{start}$;
while $OpenSet \neq \emptyset$ **do**
  remove $n_{current}$ **from** $OpenSet$;
  $n_{current} \leftarrow \underset{n \in OpenSet}{\arg\min}\ f[n]$;
  if $n_{current} = n_{goal}$ **then**
    return $TraveresedPath[n_{current}]$;
  end
  foreach $n_{neighbour} \in$ NeighboursOf$(n_{current})$ **do**
    if $g[n_{neighbour}] < g[n_{current}] +$ EdgeCost$(n_{current}, n_{neighbour})$ ***and***
    $n_{neighbour} \in OpenSet$ **then**
      $TraversedPath[n_{neighbour}] \leftarrow n_{current}$;
      $g[n_{neighbour}] \leftarrow g[n_{current}] +$ EdgeCost$(n_{current}, n_{neighbour})$;
      $f[n_{neighbour}] \leftarrow g[n_{neighbour}] + h(n_{neighbour})$;
      $OpenSet \leftarrow n_{neighbour}$;
    end
  end
end
return $NoPathFound$;

Algorithm 2: pseudo-code demonstrating the A* algorithm

robot discovers new obstacles in the arena. It is therefore of importance that the computed path to the goal is updated as new information about the environment is received, in other terms, a replanning step is necessary as the graph to be searched through is altered. The previously introduced A* algorithm provides no solution for the replanning problem other than re-initiating the graph search from scratch. D*Lite [34], Focused D*Lite [35] and *D*Lite* [36] have been the suggested incremental algorithms that extend A* with replanning capability. These algorithms use very similar approaches, but *D*Lite* has gained more popularity due to its simplicity and improved performance [36] [37]. This introduction mainly focuses on *D*Lite* however the fundamental idea generally applies to rest of the approaches.

D*Lite starts the search in a *backwards* manner, i.e. from the goal state towards the start state. After the initial optimal path is found, in the event of changes in the graph, the

states affected by the change will have their costs updated. The backwards approach in D*Lite makes it possible to re-use parts of the path as the robot moves and the n_start is changed. In a forward search, with every movement of the robot, the costs for the entire states would have to be updated.

D* and D*Lite perform efficiently when the changes in the graph occur in an area close the current position of the robot. On the other hand, if these variations take place further away, in some cases the A* can in fact out-perform these alternatives. Moreover, in completely unknown areas, the backward search used in D* and D*Lite prove to be less efficient than a forward search approach. More detailed information about D*Lite and an evaluation of its performance can be found in [36] and [37].

A.2.3 Any-time Algorithms

Where the planning problem gets more complicated, the introduced algorithms might fail to produce a solution in the limited allotted execution time. Any-time algorithms refer to algorithms which generate a fast and normally sub-optimal solution and further improve that solution the more time they are given. There have been several proposed methods that incorporate this concept with the aforementioned algorithms. One of the most prevalent of these algorithms is the extension of D*Lite called *Any-time* D*Lite [38]. This approach was used in [12] to perform a graph search through a *state-lattice* (see A.6). The efficiency of *Any-time* D*Lite can be further improved by using the modifications in D*Lite [37].

A.3 Randomized Sampling Based Methods

Sampling based algorithms are a class of algorithms mainly characterized by the property that they do not require explicit information about the obstacles in the configuration space a priori. Instead they check the feasibility of trajectories generated to connect a set of randomly sampled configurations. Hence, the solution will be guaranteed to be collision-free. These algorithms are not complete in the sense described in A.2. However, they are *probabilistically complete*, i.e. the probability of finding a solution if one exists, converges to one as the number of samples approaches infinity [6].

A.3.1 Probabilistic Road Maps

The idea of motion planning by means of randomly sampling the state-space first emerged in 1996 in a frame-work known as *probabilistic roadmap* (PRM) [39]. PRM is a *multiple-query* algorithm in the sense that it pre-computes a graph of obstacle-free trajectories which is then used to return the least costly path between the queried start and final

configurations. This approach has proven effective in a structured setting with static obstacles, e.g. off-line planning of industrial manipulators functioning in an static environment.

A.3.2 Rapidly-exploring Random Trees (RRT)

Another member of this class of algorithms, geared towards systems under kinodynamic constraints and operating in dynamically changing environments is known as *Rapidly-exploring random trees* or RRT, which was introduced in 2001 [40]. Being an incremental algorithm, in contrast to PRM, RRT does not require pre-computing a graph with a certain number of samples. Instead, it returns a solution when a valid path consisting of the required number of trajectories is computed. This algorithm has also the *any-time* property i.e. it can return an initial solution which is improved given more time. RRT is summarized in algorithm 3. After initializing the RRT graph, function SAMPLE takes a random sample x_{random} from the configuration space. Function GETCLOSESTVERTEX then finds the closest vertex $x_{closest}$ to the newly sampled configuration. Afterwards NEWEDGE constructs an edge between $x_{closest}$ and x_{random}. If this edge is collision-free, the RRT graph is updated with the new vertex and edge.

Data: initial state x_{start} and goal state x_{goal}
Result: $G = (V,E)$
while x_{goal} *not reached* **do**
  $G \leftarrow (x_{start},\emptyset)$;
  $x_{random} \leftarrow$ SAMPLE() ;
  $x_{closest} \leftarrow$ GETCLOSESTVERTEX(V,x_{random}) ;
  $edge \leftarrow$ NEWEDGE($x_{closest}$,x_{random}) ;
  if ISCOLLISIONFREE($edge$) **then**
    $V \leftarrow V \cup x_{random}$;
    $E \leftarrow E \cup edge$;
  end
end
return $G \leftarrow (V,E)$;

Algorithm 3: *RRT* algorithm [6]. See text for function descriptions.

A.3.3 RRT*

RRT* is the *asymptotically optimal* extension of RRT. Asymptotic optimality means that the solution will converge to an optimal one if the number of samples approaches infinity. The complexity of RRT* is however still closely comparable to that of RRT [6].

Algorithm 4 summarizes the steps in RRT*. The algorithm shares the same initializing

and sampling of RRT. However, after acquiring the random sample x_{random}, a set of nearby vertices X_{near} is found by function GetCloseVertices. Then some temporary edges are created between the random sample and the vertices in $x_{near} \in X_{near}$. The cost of the edges that are collision-free is calculated by EdgeCost and added to the path cost up until the corresponding x_{near} returned by PathCost. The x_{near} that has the lowest sum of these tow costs is then added to the RRT* graph along with x_{random} and the edge connecting the two which we refer to as $edge_{new}$. Next, the possible changes occurred in the previous step must be propagated throughout the rest of the graph. In order to achieve this, first the vertices contained in X_{near}/x_{random} are connected to the x_{random} (which is now newly connected to the rest of the graph via $edge_{new}$) by temporary edges *edge* and checked for being collision-free. If the path cost until an $x_{near}$is larger than the cost until x_{random} plus the edge cost of the corresponding *edge*, then the last edge leading to that x_{near} is removed from the graph. This step ensures that the less costly paths to the first-degree vertices via the newly added x_{random} replace the existing paths. In 4, this step is denoted by the function PropagateGraphChanges.

Data: initial state x_{start} and goal state x_{goal}
Result: $G = (V,E)$
while x_{goal} *not reached* **do**
  $G \leftarrow (x_{start},\emptyset)$;
  $x_{random} \leftarrow$ Sample() ;
  $X_{near} \leftarrow$ GetCloseVertices(x_{random}) ;
  $x_{min} \leftarrow$
  $\underset{x_{near} \in X_{near} | \text{IsCollisionFree}(edge \leftarrow \text{NewEdge}(x_{near},x_{random}))}{\arg\min} \{\text{PathCost}(x_{near}) + \text{EdgeCost}(edge)\}$;
  $G \leftarrow (V \cup x_{random}, E \cup \text{NewEdge}(x_{min},x_{random})$;
  $G \leftarrow$ PropagateGraphChanges(G) ;
end
return G ;

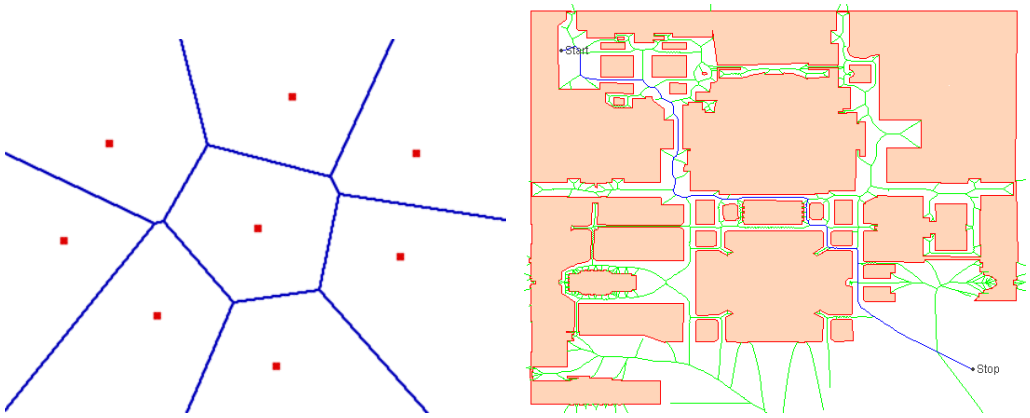
Algorithm 4: *RRT** algorithm [6]. See text for function descriptions.

A.4 Geometric Methods

A.4.1 Voronoi Diagram

The Voronoi diagram was first introduced in 1908 by Georgy Voronoi. Such a diagram demonstrates the distances between sets of points in a multi-dimensional space. In the area of path planning for mobile robots, a two dimensional space is used. The diagram is formed by connecting lines equidistant to the closest pair of points in the given set (see figure A.1(a)). In a path planning problem, the polygonal obstacles are represented by a

set of points which are known a priori. This set is used to construct a Voronoi diagram which is then pruned to remove the edges colliding with the obstacles or leading to dead-ends. The resulting graph is then searched to find the shortest path to the goal. A variety of graph search methods, e.g. those mentioned in A.2, can be used in this step. Figure A.1(b) displays a pruned Voronoi diagram built in an arena and the resulting path.



(a) (b)

Figure A.1: Generalized Voronoi diagram for a series of points A.1(a) and application of the diagram in a path planning problem A.1(b). Image taken from [41]. Green lines represent the Voronoi diagram after pruning and the blue line is the path found after the search step.

Being a purely geometric method, Voronoi diagram disregards the kinodynamics of the robot and therefore cannot provide a feasible solution to a path planning problem on its own. The result is therefore normally incorporated in multi-step solutions. As seen in [32], a Voronoi diagram is used as part of the heuristic for an A* graph search. Additionally [13] uses a Voronoi diagram to construct a field around the obstacles.

A.5 Potential Fields

The idea of obstacle avoidance and path planning using potential fields was initially introduced in 1986 [42]. In this method, the agent (the robot) is modelled as a particle affected by attracting and repulsive potentials. Obstacles can be represented as repelling fields $U_{repulsive}$ while the goal can be an attracting potential $U_{attracting}$. The force $\vec{\mathbf{F}}$, acting on the particle will be:

$$\vec{\mathbf{F}} = -\vec{\nabla}(U_{repulsive} + U_{attracting})$$

In other terms, an obstacle-free path to the goal can be found by moving the particle in

the direction of $\vec{\mathbf{F}}$ i.e. descending along the gradient of the sum of the potentials. Figure A.2 illustrates this concept.

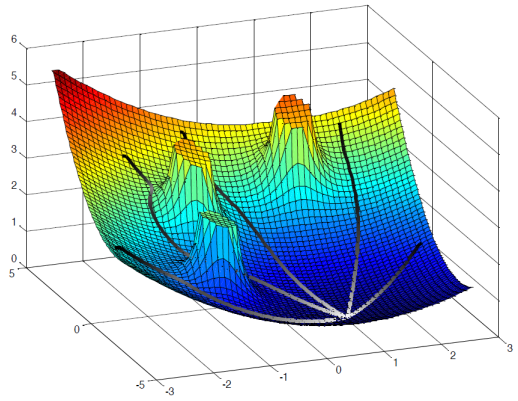


Figure A.2: a potential field consisting of three obstacles and and a goal at (1,-2). Image taken from [43]

As an example, the potentials $U_{repulsive}$ and $U_{attracting}$ can be defined as the following [42]:

$$\begin{aligned} U_{attracting} &= \frac{k|x-x_g|^2}{2} \\ U_{repulsive} &= \begin{cases} \frac{1}{2}\eta(\frac{1}{\rho} - \frac{1}{\rho_0})^2, & \text{if } \rho \leq \rho_0 \\ 0, & \text{if } \rho > \rho_0 \end{cases} \end{aligned}$$

where x is the current position of the robot and x_g the attraction point, i.e. the goal. The current shortest distance between the robot and the obstacle is denoted by ρ, while ρ_0 represents the minimum influencing distance of the potential field. k and η are constants. Other formulations of the potentials tailored to specific path planning problems have been suggested in the literature. For instance, [20] proposes a set of repulsive functions to account for lane markings, road barriers and other vehicles in a highway setting. The fields corresponding to the other vehicles change size based on the relative speed to assure a safe distance.

Being an inherently gradient based method, solutions built solely on potential fields run the risk of entrapment in local minima. This is the major issue in this approach and therefore, in the context of path planning for automotive applications, this method is typically used in conjunction with other methods. Another possible problem is in the more complex obstacle configurations where narrow passages may be assigned high potential values which makes precise manoeuvres difficult. An example of addressing this issue can be seen in [13] where, based on the geometry of the obstacle, a variable-size potential is used.

A.6 State Lattice

State-lattice was initially proposed in 2005 [44] as a solution to the path planning problem. In this approach, the state-space is discretized and a set of trajectories are calculated which connect each of the discrete states to a set of its reachable neighbours, thus forming a lattice. The result is a regular and repeatable directed graph where each vertex is a reachable state and each edge is a feasible trajectory which accounts for the motion constraints of the mobile robot. Figure A.3 displays a simple state-lattice.

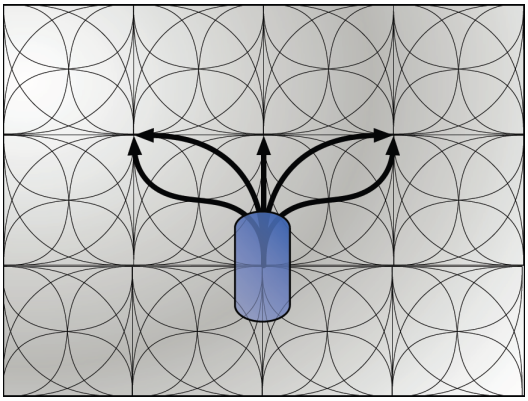


Figure A.3: a simple state-lattice. Image taken from [21]

The trajectories (graph edges) are typically calculated using relatively high-fidelity kinematic models in order to satisfy differential constraints. The regularly sampled state-space makes it possible to either pre-compute and save the connecting trajectories or generate them on-line by means of solving simpler optimal control problems. A proposed method suitable for on-line generation of trajectories can be found in [3]. This approach will be investigated in detail later on in the report (see 3.3).

Given the state-lattice, the solution for the desired navigation can be found on-line via a graph search step which satisfies the spatial constraints, i.e. an obstacle-free path. Being a directed graph, a variety of search methods such as the ones summarized in A.2 can be used.

An implementation of a lattice-based approach can be seen in [5] where the path planner was tested in a planetary rover. Another example is found in [12] where this method is applied in the DARPA Urban Challenge.

A.7 Highway Specific Methods

The approaches summarized so far mainly consider unstructured environments and try to tackle the path planning problem in its rather general form. However, more efficient and

to some extent add-hoc approaches can be formed in a structured setting with *a priori* knowledge of the characteristics of the environment as well as the expected manoeuvres.

In particular, autonomous driving in a high-way setting has been appealing to both researchers and the industry. Intelligent Vehicle Highway Systems (IVHS) has been the topic of extensive research and a variety of algorithms have been proposed for applications such as lane following, lane change, overtake and platooning. Suited for automotive applications, passenger comfort criteria such as disturbances caused by lateral and longitudinal accelerations play an important role in the design of such solutions. This is mostly overlooked in the methods developed for mobile robots intended for non-passenger applications.

B

Trajectories for Highway Manoeuvres

This appendix includes a motivation for the cost functional choice in 3.2. It is shown why minimizing jerk over a trajectory should give the best result in terms of passenger comfort. Details on how constraints on acceleration are included in the trajectory generation for highway manoeuvres are also given. For this purpose, an algorithm for iterative adjustment of the trajectory is presented.

B.1 A remark on jerk-minimized trajectories

As mentioned earlier in this report, the proposed method for the highway overtake case is based on generation of a series of jerk-minimized trajectories. The literature on the topic of trajectory planning, often regard minimizing the third time derivative of displacement as a suitable approach for increasing comfort and smoothness.

As demonstrated in [45] and [46], a trajectory $x(t)$ will connect a start point to an end point smoothly if it minimizes the following cost function:

$$J = \int_0^{t_f} \dddot{x}(t)^2 dt \tag{B.1}$$

Minimizing B.1 implies minimizing changes in force. The question might arise that why in particular minimizing the third time derivative of position results into better behaving

trajectories than when higher or lower order time derivatives are minimized. In [47], the authors study the behaviour of $x(t)$ when minimizing the following cost functional:

$$J = \int_0^{t_f} (d^n x(t)/dt^n)^2 dt \tag{B.2}$$

As the derivation order n increases, the maximum speed over the trajectory $x(t)$ will also increase. Figure B.1 displays $x(t)$ and the speed along the trajectory when B.2 is minimized for $n \in \{2,3,4,5,6\}$. The 4th, 5th and 6th time derivatives of $x(t)$ i.e. $n \in \{4,5,6\}$ are normally referred to as snap, crackle and pop respectively. These results were obtained numerically by formulating the corresponding optimal control problem using the cost function B.2 subject to the differnetial constraints and boundry conditions in B.3. The problems were solved using BOCOP — an open-source optimal control solver, for $t_f = 4$ seconds and $x_{t_f} = 4$ meters.

$$\begin{aligned}
\begin{bmatrix} \dot{x}(t) & \dot{x}^{(1)}(t) & \cdots & \dot{x}^{(n-1)}(t) & \dot{x}^{(n)}(t) \end{bmatrix} &= \begin{bmatrix} x^{(1)}(t) & x^{(2)}(t) & \cdots & x^{(n)}(t) & u(t) \end{bmatrix} \\
\begin{bmatrix} x(0) & x^{(1)}(0) & \cdots & x^{(n)}(0) \end{bmatrix} &= 0 \\
x(t_f) &= x_{t_f} \\
\begin{bmatrix} x^{(1)}(t_f) & \cdots & x^{(n)}(t_f) \end{bmatrix} &= 0
\end{aligned} \tag{B.3}$$

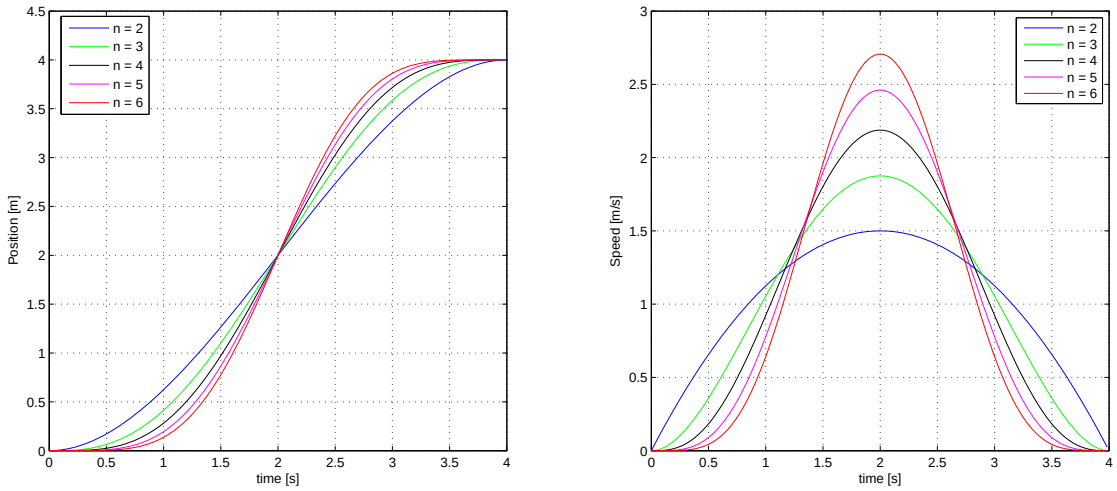


Figure B.1: $x(t)$ (left) and its first derivative (right) when the cost function B.2 is minimized for different values of n subject to constraints in B.3

If r denotes the ratio between the maximum speed along $x(t)$ to the speed averaged along the trajectory [47] [48], table B.1 shows the values for r obtained for the corresponding

Table B.1: Values calculated for r when $n \in \{2,3,4,5,6\}$

derivation order n	ratio r
2	1.5000
3	1.8750
4	2.1875
5	2.4609
6	2.7070

cases of acceleration-, jerk-, snap-, crackle- and pop-minimized trajectories displayed in figure B.1.

Through experiments [45], it has been concluded that the ratio r for a so called reaching movement performed by human beings has an approximated value of 1.8. As shown earlier, r is closest to this value when $n = 3$ in other words when jerk is minimized. These observations have been of particular interest in the area of brain and cognitive science, computational neurobiology and rehabilitation [49].

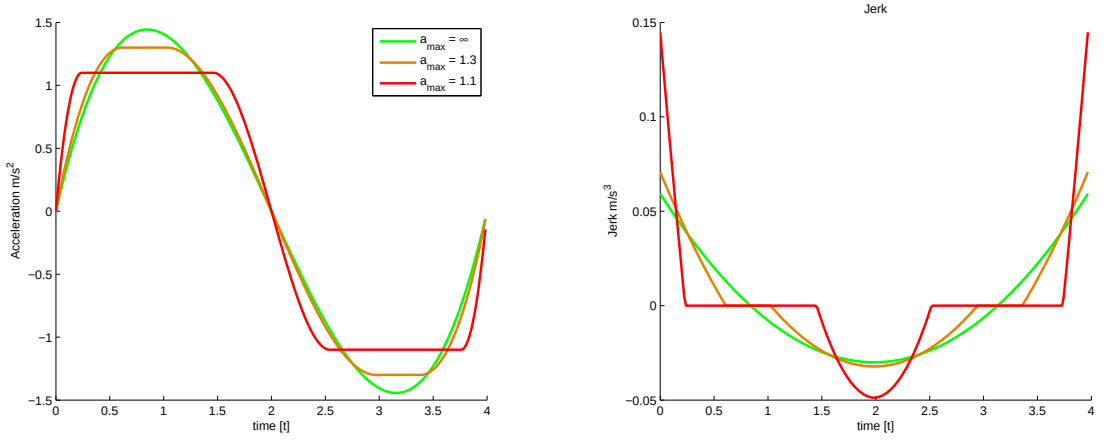
The effect of jerk on perception of motion has been proven both neurophysiologically [50] and psychophysically [51] [52] [53]. With this said, criteria for passenger comfort under normal driving conditions in ground vehicles have rarely been a topic of investigation. Generally, the changes of motion felt by the passengers determines their comfort [54]. Jerk as a comfort criteria has been studied in [55] while [56] finds jerk as the most important comfort criterion. Constant speed does not impose any forces on passengers, acceleration on the other hand is felt. Constant acceleration can be tolerated by passengers by means of bracing themselves, however, variations in acceleration (jerk) is said to be the main reason of discomfort due to causing loss of balance [56].

B.2 Inclusion of state constraints

Consider a variation of problem 3.3 where symmetric restrictions on acceleration are made

$$
\begin{aligned}
\min_{u(t)} \quad & \int_0^{t_f} \frac{1}{2} u^2(t) dt \\
\text{s.t.} \quad & \dot{x}_1(t) = x_2(t) \\
& \dot{x}_2(t) = x_3(t) \\
& \dot{x}_3(t) = u(t) \\
& |x_3(t)| \leq a_{max} \\
& x_1(0) = x_{10},\ x_2(0) = x_{20},\ x_3(0) = x_{30} \\
& x_1(t_f) = x_{1f},\ x_2(t_f) = x_{2f},\ x_3(t_f) = x_{3f}
\end{aligned}
$$

Due to the hard endpoint constraints x_{1f},x_{2f}, x_{3f} it is evident that there are no general solution the problem, since feasible ones might not exist. Instead, the qualitative properties of the solution are conditioned on the current configurations of boundary conditions and final times. Assuming feasibility, the problem has either the same solution as the unrestricted problem or one where the lateral acceleration is saturated for periods of time. The principal behaviour of the system for a situation where this occurs is shown in Figure B.2, where the results are numerical and retrieved through the solver BOCOP



(a) Acceleration Profiles with and without restriction

(b) Jerk Profiles with and without restriction

Figure B.2: maps with limits

Even though a closed form solution can be found for problems of this type, there is an important relationship between maximum acceleration and jerk making it less desirable. As noted in [57] trajectories with low levels of maximum acceleration will always be coupled to higher levels of maximum jerk for any same boundary conditions. Specifically, the levels of jerk subjected to the system and the time during which it is saturated in $x_3(t)$ is strongly related. Periods of saturation, especially longer ones, necessitates higher levels of jerk during the transitions to and from the maximum acceleration levels. Letting u_r and u_u denote the jerk levels of the restricted and unrestricted problem respectively,

one can see even without mathematical rigour that given the same boundary conditions and final time the following must hold

$$\max_t|(u_u(t)|) \leq \max_t(|u_r(t)|) \tag{B.4}$$

Specifically $\max(u_r(t) \rightarrow \infty$ as the system approaches its feasibility limits, resulting in a system behaviout similar to that of *Bang-Bang*-control. Due to this a different approach has been taken focusing more on reducing the overall jerk levels than pushing the optimality limits of the system and thereby getting shorter periods of higher jerk.

The main idea is to restrict the acceleration extrema to lie in a certain interval which bounds are set to be the maximum allowed acceleration, i.e.

$$\max_{t\in(0,t_f)}(|x_3(t)|) \in [0, a_{max}]$$

so that

$$x_3(t) \in [-a_{max}, a_{max}]$$

These maxima obviously occurs at the roots of $\dot{x}_3$. By substituting these into Equation 3.4 and equating it to the specified bound on acceleration one gets a relation between the extrema and boundary conditions of the system. Considering the final time t_f as variable and solving for it, one retrieves the final time for which the trajectory given by the specified boundary conditions has its maxima exactly at the limiting value.

Formally,by differentiating 3.4 so that

$$\dot{x}_3(t) = -\frac{1}{2}C_1t^2 + C_2t - C_3$$

and setting $\dot{x}_3(t) = 0$ the time at which the extrema occur is found as

$$t_{max} = \frac{C_2}{C_1} \pm \sqrt{\left(\frac{C_2}{C_1}\right)^2 - \frac{2C_3}{C_1}} \tag{B.5}$$

The specific root of interest, at which the highest value in magnitude is achieved, is determined by C_1,C_2 and C_3, i.e, by the boundary conditions. Now, setting the acceleration at the time t_{max} to the prescribed maximum value one gets

$$a_{max} = -\frac{1}{6}C_1t_{max}^3 + \frac{1}{2}C_2t_{max}^2 - C_3t_{max} + C_4 \tag{B.6}$$

After substituting B.5 and 3.7-3.12 one gets a 13:th order polynomial in the final time t_f, i.e. an equation that lacks analytic solution. However, in the special case where the particle moves from one stationary point to another, i.e. $x_{20} = x_{30} = x_{2f} = x_{3f} = 0$ it can be shown that

$$t_f^M = \sqrt{\frac{10}{\sqrt{3}}\frac{1}{a_{max}}|(x_{1f} - x_{10})|} \tag{B.7}$$

Where t_f^M then is the final time for which the maximum acceleration of the trajectory is equal to that of the bound a_{max}. Now, since this situation only will be of interest at the time instance where a manoeuvre is commenced, and not be applicable for any in-between replanning situation. The solution of Equation B.6 must therefore be found numerically for all non stationary boundary conditions. Algorithm 5 based on interval halving has been implemented that solves this. Assuming that the maxima of the sought trajectory at the given non stationary boundary conditions together with the nominal final time yields maximal accelerations that in magnitude are greater than the specified bound, the actual final time for which the bounds are respected is found through the algorithm by iterative adjustment of the nominal final time.

An example of the progress for a typical replanning situation,i.e., planning when an old trajectory already is traversed, with non stationary boundary conditions is shown in Figure B.3, where the procedure is visualized.

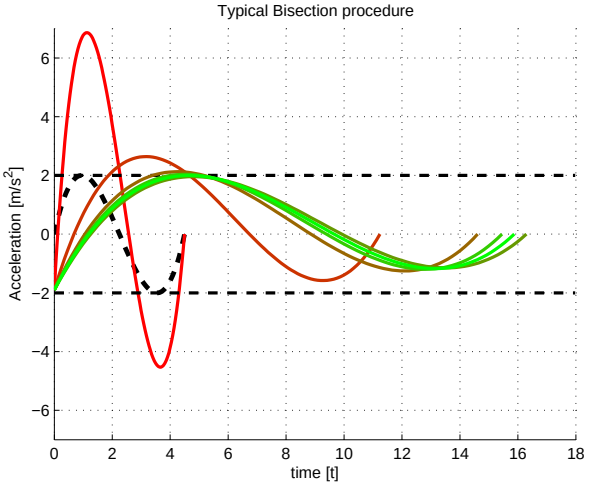


Figure B.3: Proceedings of Algorithm 5 when with the boundary conditions $x_{10} = 0$,$x_{20} = -5, x_{30} = -1.9, x_{1f} = 7, x_{2f} = 0$ and $x_{3f} = 0$. a_{max} is set to 2 and and visualized as dotted horizontal lines. The nominal curve, computed with stationary boundary conditions are shown as the dashed curve. The value at the iterations are shown as colored lines, where greener hues are later iterates and red earlier.

Very few iterations are normally needed and the execution time is negligible. The procedure GetMaximumAcceleration(t) returns the acceleration largest in value within the range (0,t). Note that this assumes that initial and final conditions on acceleration are within the specified bounds. GetNominalTime(B,a_{max}) simply returns the value of Equation B.7. By using this procedure, all generated trajectories can therefore be guaranteed to have accelerations within specified border while at the same time minimizing the cost measure defined in Equation 3.3.

Data: Boundary Conditions B,Acceleration Bound a_{max}
Result: final time t_f
$t_f^{nom} \leftarrow$ GetNominalTime(B,a_{max});
$a_{nom} \leftarrow$ GetMaximumAcceleration(t_f^{nom});
if $|a_{nom}| < a_{max}$ **then**
  return t_f^{nom}
else
  $f_{low} \leftarrow |a_{nom}| - a_{max}$;
  $t_f^{low} \leftarrow t_f^{nom}$;
end
$P \leftarrow 2$;
while $f_{high} > 0$ **do**
  $t_f^{high} \leftarrow t_f^{low} P$;
  $a_{high} \leftarrow$ GetMaximumAcceleration(t_f^{high});
  $f_{high} \leftarrow |a_{high}| - a_{max}$;
  $P \leftarrow P + 1$;
end
$k \leftarrow 0$;
while $k < k_{max}$ **do**
  $k \leftarrow k + 1$;
  $t_f^{mid} \leftarrow \frac{(t_f^{high} + t_f^{low})}{2}$;
  $a_{mid} \leftarrow$ GetMaximumAcceleration(t_f^{mid});
  $f_{mid} \leftarrow |a_{mid}| - a_{max}$;
  if $|f_{mid}| < \varepsilon$ ***or*** $(t_{high} - t_{low})/2 < \varepsilon$ **then**
    return t_f^{mid}
  else
    if $sign(f_{mid}) = (f_{low})$ **then**
      $f_{low} \leftarrow f_{mid}$;
      $t_f^{low} \leftarrow t_f^{mid}$;
    else
      $f_{high} \leftarrow f_{mid}$;
      $t_f^{high} \leftarrow t_f^{mid}$;
    end
  end
end
return t_f^{mid}

Algorithm 5: Bisection Algorithm for finding final time t_f

C

Cost Components

This Appendix details the different cost components introduced in 3.4.1 and used in the trajectory selection. The reader should note that the costs presented only are proposed *alternatives* and that many others could be considered.

Jerk Following the reasoning in 2.3 which will be discussed further in 3.2 the squared integral of lateral jerk has been selected as a measure of comfort. Formally, if the unscaled cost of one trajectory t_j is given as

$$J_j = \int_{t_j} j_{lat}^2 ds$$

where the integral is taken over the spanned trajectory. The output of the associated cost function is then formed as

$$f_{jerk}(T,t_j,) = -\frac{J_j}{\max\limits_i(J_i)}$$

where the minus sign is included since high jerk levels are a negative property. The intention with this definition is to give a inclination towards those trajectories exhibiting low overall jerk levels.

Maximal Acceleration To get at preference for trajectories with low maximal lateral acceleration the following cost is defined for each trajectory t_j

$$A_j = \max_{t_j}(|a_{lat}|)$$

where the max operator is taken over the entire trajectory. The normalized output is then formed according to

$$f_{acc}(T,t_j) = -\frac{A_j}{\max_i(A_i)}$$

The minus sign is again included due to the adverse effects of the described property

Deviation from desired Lane Since it in most cases exist one lane is more desireable to be positioned in a certain lane, e.g. the rightmost lane during normal operation, left lane in preparation for a left turn etc., a cost is introduced to "pull" the host towards it. Letting y be the lateral position of the current trajectory endpoint, y_d the lateral position of the desired lane center and W_R the total road width, the measure is formed as

$$f_{lane}(t_j,E) = -\frac{|y - y_d|}{W_R}$$

Deviation from Lane Center In many cases it is necessary to traverse the road in other lanes than desired one, e.g. during overtaking. It is however in that case motivated by safety, comfort and traffic regulations to be positioned close to the current lane center. A punishment from deviating from the closest lane center is therefore added. In a road bound coordinate system with its horizontal axis along the centreline, the measure is taken as

$$f_{center} = \frac{\cos\left(2\pi\frac{2y - W_L(N-1)}{2W_L}\right) - 1}{2}$$

visualized in Figure C.1. Here, W_L,N are the width of a lane and number of lanes respectively, while y is the lateral coordinate.

Terminal Longitudinal Velocity To premier higher velocities and thereby faster road traversal a speed dependent component is introduced. Since the longitudinal control is outside the reach of the planner, lateral motions letting the ACC works in its set speed adapting mode will thereby score higher. Denoting the terminal velocity of trajectory j $v_{f,j}$ and the ACC set speed v_S, the measure is formed as

$$f_{vel} = \frac{v_{f,j}}{v_s} \tag{C.1}$$

Note that this is the only positive contribution presented in this section. Without it, or some other cost with similar impact, the planner will eventually "get stuck" laterally as the vehicle rather will slow down and stay in- rather than manoeuvre away from the desired lane once it is reached. Equation C.1 will therefore "pull" the vehicle around obstacles along faster trajectories such are available.

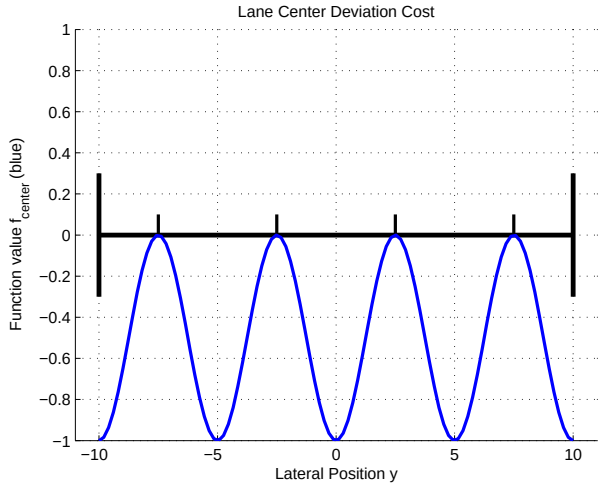


Figure C.1: Cost of lane center deviation. The blue line is f_{center} while the thinner black lines marks the lane centres.

Proximity to Obstacles Heavily related to safety, the most important criteria against which all trajectories must be validated is that of the proximity to nearby obstacles. Since the trajectories extend forwards in time, this measure must be computed using the prediction of obstacle and host vehicle motion over some horizon τ_f. Refraining from a precise definition of how it is obtained, the set of predicted obstacle trajectories is denoted O, containing individual elements $o_l, l = 1,...,|O|$. The distance to other road occupants considered dangerous or uncomfortable is highly asymmetric in the road coordinates (x,y) due to which the standard euclidean norm is ill suited as a basis for the desired measure. Compare for example the smaller distance at which it is perceived as safe to pass a vehicle with the headway distance used during normal highway driving. The asymmetries essentially raises the need for two distance measures; one for lateral displacement and one for longitudinal. Capturing both demands, a dynamically varying weighted norm is therefore defined according to

$$g(t_j,o_l,\tau) = \sqrt{\frac{\Delta x^2(\tau)}{\sigma_x^2(v_{o_l}(\tau),v_{t_j}(\tau))} + \frac{\Delta y^2(\tau)}{\sigma_y^2}}$$

where v_{t_j} and v_{o_l} are the longitudinal velocities along the trajectories of the host and obstacle respectively and $\Delta y(\tau) = t_{jy}(\tau) - o_{ly}(\tau)$, $\Delta x(\tau) = t_{jx}(\tau) - o_{lx}(\tau)$. The variables $t_{jx}(\tau)$,$o_{lx}(\tau)$,$t_{jy}(\tau)$ and $o_{ly}(\tau)$ are here meant to be read as the (x,y) coordinates at time τ. Geometrically, $g(t_j,o_l,\tau)$ is an elliptic function with minor and major axes determined by $\sigma_x(v_{o_l}(\tau),v_{t_j}(\tau))$ and σ_y, where the former is time (or rather speed) dependent. Although somewhat an abuse of notation, the arguments t_j and o_l should thereby be interpreted as multidimensional collections containing descriptions of the relevant predicted states of the host and current obstacle respectively. To allow a bounded numerical

range and functional values increasing with proximity, $g(t_j,o_l,\tau) \in \mathbb{R}$ is restricted to the unit interval by

$$\hat{g}(t_j,o_l,\tau) = \frac{1}{1+g(t_j,o_l,\tau)} \tag{C.2}$$

While keeping the elliptic shape, this function reaches it's maximum $\hat{g}(t_j,o_l,\tau) = 1$ if the host and obstacle trajectories intersect in (x,y,τ), and decays towards 0 as they move apart. However, $\hat{g}(t_j,o_l,\tau)$ lacks physical meaning rendering it unintuitive and therefore complicated to specify limits and evolution properties of the measure. Letting $c \in (0,1)$ be the limiting value of $\hat{g}(t_j,o_l,\tau)$, i.e. the value at which a proximity limit is violated, and defining Δx_{lim} and Δy_{lim} as the distances in the purely longitudinal (x) and purely lateral (y) dimension at which this is achieved, one gets with Equation C.2 that

$$\hat{g}(t_j,o_l,\tau) = \frac{1}{1+g(t_j,o_l,\tau)} = c \;\rightarrow\; \sqrt{\frac{\Delta x_{lim}^2}{\sigma_x^2(v_{o_l}(\tau),v_{t_j}(\tau))} + \frac{\Delta y_{lim}^2}{\sigma_y^2}} = 1/c - 1$$

Letting the longitudinal distance $\Delta x = 0$, σ_y can then be found as

$$\frac{\Delta y_{lim}}{\sigma_y} = 1/c - 1 \;\rightarrow\; \sigma_y = \frac{\Delta y_{lim}}{1/c-1}$$

Similarly, setting $\Delta y = 0$ gives σ_x as

$$\sigma_x(v_{o_l}(\tau),v_{t_j}(\tau)) = \frac{\Delta x_{lim}(v_{o_l}(\tau),v_{t_j}(\tau))}{1/c-1}$$

In this way all values $\hat{g}(t_j,o_l,\tau) > c$ will correspond to relative, asymmetric distances below the specified limit, i.e. with a proximity deemed to be to high. The choice of c can in principle be set arbitrary as long as $c \in (0,1)$. However, by changing c one affects the growth rate, i.e. derivative of $\hat{g}$ and thereby the characteristics of the measure in the allowed region. As seen demonstrated in Figure C.2 it tends to be more linear in the vicinity of the limiting distance as $c \to 1$ and contrary heavily non-linear as $c \to 0$

Besides the previously treated asymmetries in (x,y), an additional one exist in the longitudinal direction. Depending on the relative speed between the host and obstacle, the proximity measure can be differentiated for $\Delta x > 0$ and $\Delta x < 0$. Consider for example a case where the host is overtaking a slower moving obstacle. Clearly, the limiting distance needed in the beginning of the manoeuvre when the host is *behind* the obstacle is greater than that when it returns to the lane *in front* of it. A remedy for this is to further augment the function used, and introduce different fall-off rates σ_x for the front and rear end of the obstacle. One intuitive way to realize this is to introduce logical conditioning, i.e. *if-then-else* branching, selecting between cases. However, to ensure continuity and promote simplicity a smooth blending is preferred and can be set up as

$$p(t_j,o_l,\tau) = S(\Delta x(\tau))\hat{g}_1(t_j,o_l,\tau) + S(-\Delta x(\tau)))\hat{g}_2(t_j,o_l,\tau) \tag{C.3}$$

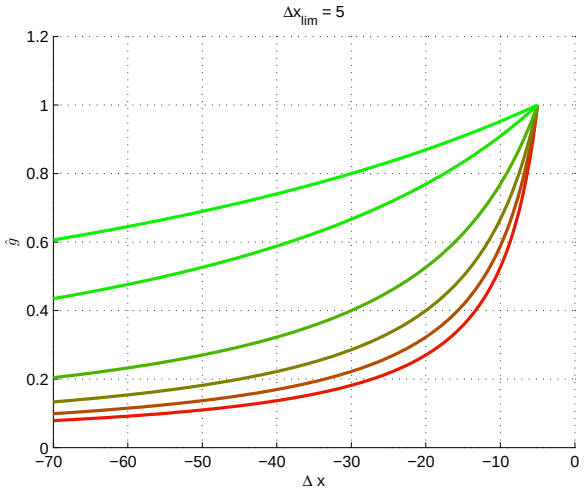


Figure C.2: Behaviour of $\hat{g}(t_j,o_l,\tau)$ close to some limiting distance. The red hues correspond to small values of c and green to higher. Note that $\hat{g}(t_j,o_l,\tau)$ here is normalized with c for easier comparison.

if $S(x)$ is a bounded function such that $S(x) \to d$ as $x \to \infty$, $S(x) \to 0$ as $x \to -\infty$ and $S(x) + S(-x) = d$, $\forall x$, given some $d \in \mathbb{R}$. The functions $\hat{g}_1(t_j,o_l,\tau)$ and $\hat{g}_2(t_j,o_l,\tau)$ are defined as in Equation C.2 but differentiated by replacement of σ_x with σ_{x1} and σ_{x2} respectively. The requirements on $S(x)$ are met by the (logistic) sigmoid function

$$S(x) = \frac{1}{1 + e^{-\beta t}}$$

since

$$S(x) + S(-x) = \frac{1}{1 + e^{-\beta t}} + \frac{1}{1 + e^{\beta t}} = \frac{2 + e^{-\beta x} + e^{\beta x}}{(1 + e^{-\beta x})\,(1 + e^{\beta x})} = \frac{2 + e^{-\beta x} + e^{\beta x}}{2 + e^{-\beta x} + e^{\beta x}} = 1$$

and

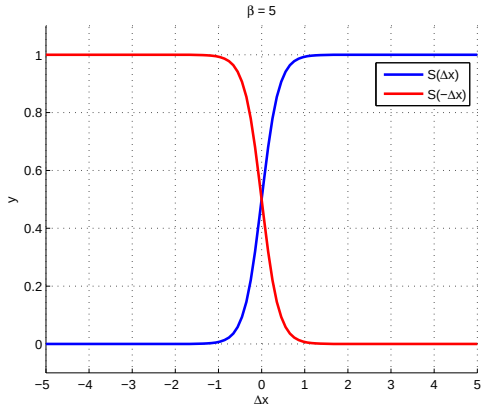
$$\frac{1}{1 + e^{-\beta t}} \to 1 \quad \text{as} \quad x \to \infty, \qquad \frac{1}{1 + e^{-\beta t}} \to 0 \quad \text{as} \quad x \to -\infty$$

Here, β determines the "slope" of the nearly linear region. The principal behaviour of the blending is shown in Figure C.3

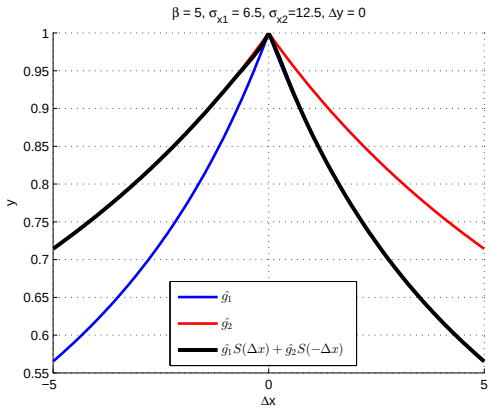
The necessary distance limits Δy_{lim} and Δx_{lim} and thereby $\sigma_{x1}, \sigma_{x2}, \sigma_y$ are by nature tuning parameters and deeper examination on suitable choices are outside the scope of this thesis. Some values can however be derived from road geometry and simple dynamic relationships. For instance, a simple choice for the lateral dimension is

$$\Delta y_{lim} = \sigma_y(1/c - 1) = W_L/2$$

where W_L is the lane width. Further, involving speed dependence, two basic measures can be used to determine the limiting distance in the longitudinal direction. Firstly, the



(a) Evolution of the logistic function with positive and negative arguments



(b) Cross-section of the proximity measure given from combining the two elliptic functions as defined in Equation C.3

Figure C.3: Principal behaviour of the blending functions and the resulting proximity measure. Numerical details are found in the figure headers.

relative velocity between host and obstacle together with some specified headway time τ_h can be used to form

$$\Delta x^I_{lim} = (v_{o_l}(\tau) - v_{t_j}(\tau))\tau_h$$

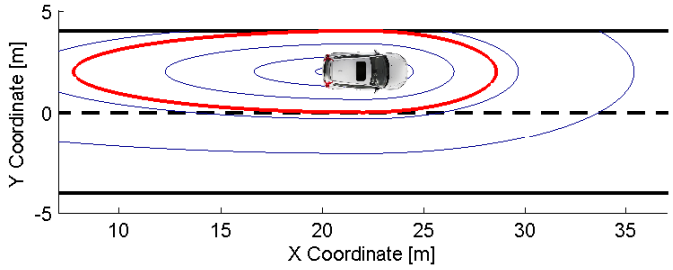
Secondly, a dependence on the absolute speed of the host vehicle can be included as

$$\Delta x^{II}_{lim} = D + v_{t_j} k$$

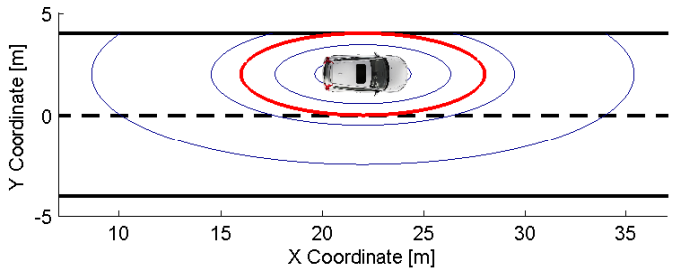
for some constants k and D. The latter effectively forms the minimum distance allowed. The limiting measures are then taken as

$$\begin{aligned} \sigma_{x1} &= \frac{\max(\Delta x^I_{lim} + \Delta x^{II}_{lim}, \Delta x^{II}_{lim})}{1/c - 1} \\ \sigma_{x2} &= \frac{\max(-\Delta x^I_{lim} + \Delta x^{II}_{lim}, \Delta x^{II}_{lim})}{1/c - 1} \end{aligned}$$

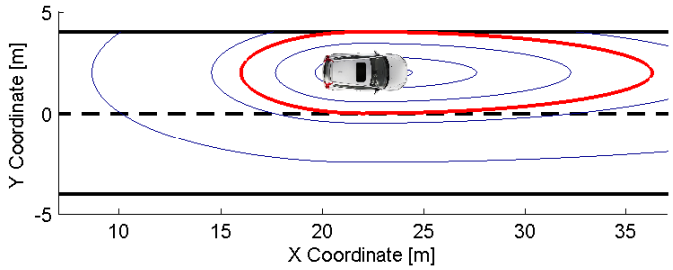
The behaviour of the function using these rules can be seen in Figure C.4



(a) $v_{o_l} - v_{t_j} < 0$



(b) $v_{o_l} - v_{t_j} = 0$



(c) $v_{o_l} - v_{t_j} > 0$

Figure C.4: Visualization of the proximity measure at some time τ. The red lines indicate the contour line $p(t_j,o_l,\tau) = c$ and the blue lines some other levels curves of the function. With the notation used above the following values were used: $\beta = 5$, $c = 0.5$, $k = 0.1$, $D = 5$, $W_L = 4$

Finally, the proximity cost is taken as the maximal proximity over the prediction of the currently evaluated trajectory, considering *all* obstacles and normalized with the highest allowed functional value c. Formally, taking τ_f as the current prediction horizon and letting

$$M(t_j,O) = \max_{o_l \in O} \left(\max_{\tau \in (0,\tau_f)} (p(t_j,o_l,\tau)) \right)$$

the measure is defined according to

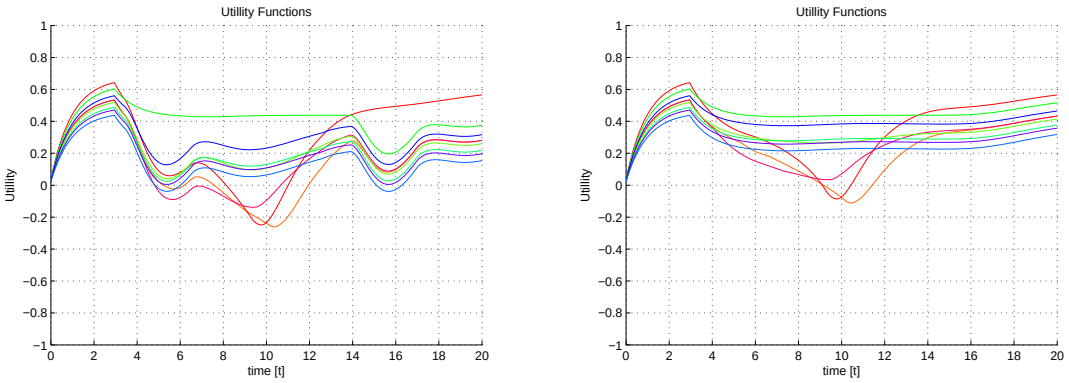
$$f_{prox} = \begin{cases} -\frac{M(t_j,O)}{c}, & \text{if} \quad M(t_j,O) > c \\ -\infty, & \text{otherwise} \end{cases} \tag{C.4}$$

where the second statement will indicate that the current trajectory should be discarded for violating set proximity limits. By this $f_{prox} \in (-1,0)$ aligning it with the other costs defined above. In practice, the cost is evaluated on a discrete set of points along the trajectory and the maximum is approximated by the highest values at these.

Stationarity As mentioned above and detailed in 3.4, discrete systems with ill designed selection procedures might in some cases produce the chattering phenomenon. Even though a filtered measure indeed reduces the risk, it is by no means guaranteed to vanish. To increase safety margins and thereby further increase robustness, a contribution is introduced intended to premier fulfilment of initiated movements. That is, once a motion has commenced, a penalty is placed on switching. With $a_{lat}(t)$ as the current lateral acceleration of the vehicle, the measure is defined according to

$$f_{stat} = \begin{cases} -\frac{|a_{lat}|}{a_max} & \text{when } |a_{lat}| > \varepsilon \\ 0 & if|a_{lat}| < \varepsilon \text{ or trajectory is already active} \end{cases}$$

Here, a_{max} is a global limitation set by the planner on all generated trajectories and $\varepsilon > 0$ the limit determining when the cost is activated. Assuming that no external steering commands are given so that the $a_{lat}(t) < a_{max}\ \forall t \in (0,t_f)$ the measure is restricted to $f_{stat} \in [-1,0]$. Figure C.5 shows the effects of this during a typical lane change scenario. The costs of the trajectories not selected are clearly lowered after switching takes place, creating the desired margin.



(a) Evolution with Stationarity Cost

(b) Evolution without Stationarity Cost

Figure C.5: Principal effects of the stationarity cost. Parameters used here was $\tau = 1, \alpha = 1/3$. Active cost contributions: *Proximity*, *Deviation From Lane Center*, *Deviation from desired lane* and *Terminal Longitudinal velocity* with corresponding weights 2,1,1,4. The weight for the stationarity cost was set to 4 in (a)

D

Trajectories for Turning

This Appendix provide details relevant to the numerical trajectory generator presented in 3.3. The Levenberg Marquardt Algorithm is intrdouced and explained together with a detailed, step-by-step description of the algorithm that has been implemented. Further, a procedure with which the decision variables of the problem are scaled to augment numerical stability is detailed. A short description is also given on the *Composite Simpson Quadrature*, which is utilized for numerical integration.

D.1 Levenberg Marquardt Algorithm

Selected for the implementation of Equation 3.25 is the Levenberg-Marquardt algorithm as presented in [22], due to its frequent use in multidimensional root finding problems. In essence, it constitutes a smooth mix between first and second order methods (e.g. gradient descent and newtons method) applied to the squared residual. As the algorithm progresses its characteristics vary between the two extremes as a function of the behaviour of the objective near the current decision variable values. In essence, it takes small steps where the function is changing rapidly and large ones . Formally, this is performed by a linearisation of the function to be solved around the decision variables at the current iterate $\mathbf{q} = [d,\ s_f]$ followed by equation of the linear approximation to zero, i.e.

$$\mathbf{g}(\mathbf{q} + \Delta\mathbf{q}) \approx \mathbf{g}(\mathbf{q}) + \left.\frac{\partial \mathbf{g}}{\partial \mathbf{q}}\right|_{\mathbf{q}} \Delta\mathbf{q} = 0 \quad \Rightarrow \quad \left.\frac{\partial \mathbf{g}}{\partial \mathbf{q}}\right|_{\mathbf{q}} \Delta\mathbf{q} = -\mathbf{g}(\mathbf{q})$$

Where $\mathbf{g}(\mathbf{q}) = [g_1(\mathbf{q}),\ g_2(\mathbf{q})]$ after Equation 3.25. With the jacobian

$$J = \left.\frac{\partial \mathbf{g}}{\partial \mathbf{q}}\right|_{\mathbf{q}} \tag{D.1}$$

both sides are multiplied with $J^\top$

$$\mathrm{J}^\top \mathrm{J} \Delta \mathbf{q} = -\mathrm{J}^\top \mathbf{g}(\mathbf{q}) = -\nabla r(\mathbf{q}) \tag{D.2}$$

where $r(s,d) = \frac{1}{2}||\mathbf{g}(d,s_f)||^2$ is the squared residual of the system of equations. To introduce diagonal dominance and thereby ensure invertability, Equation D.2 is augmented to

$$\left(\mathrm{J}^\top \mathrm{J} + \lambda I\right) \Delta \mathbf{q} = -\mathrm{J}^\top \mathbf{g}(\mathbf{q}) = -\nabla \mathbf{g}(\mathbf{q}) \tag{D.3}$$

and the increment is thereby given as

$$\Delta \mathbf{q} = -\left(\mathrm{J}^\top \mathrm{J} + \lambda I\right)^{-1} \nabla \mathbf{g}(\mathbf{q}) \tag{D.4}$$

The parameter $\lambda > 0$ now governs both the direction and size of the increment. Large λ essentially turns the direction into that of gradient descent and reduces the step size. Conversely, $\lambda \to 0$ yields a large step in the direction of the approximate newtons method.

The product $\mathrm{J}^\top \mathrm{J}$ can be seen as an approximation of the Hessian of the squared residual as it is used in Newtons Method.

$$\gamma = \frac{\Delta f_Q}{\Delta f_A}$$

where $\Delta f_Q, \Delta f_A$ denotes the reduction in the quadratic model and actual function respectively. An adjustment rule is thereafter conditioned on the value of γ, usually to lower λ if $\gamma > b$ and increase if $\gamma < a$ for some $0 < a < b$. Following this the computed step is only accepted if $\gamma \in (a,b)$, i.e. the computed result for bad choices of λ are discarded and the values of the decision values kept to the next iteration. This means that if the objective is badly conditioned in the vicinity of the current iterate, several function evaluations are needed to adjust λ before actual progress can be made. To reduce the number of function evaluations, a simpler approach is utilized based on the presentation in [22]. Disregarding model fidelity, λ is reduced and a step taken if it leads to a reduction in the residual, otherwise the previous values are kept and λ increased. Consequently, λ is never "fine tuned" to the best possible value by which the needed number of function evaluation decreases. It should be noted that this choice comes with

a cost, as the method renders the convergence more sensitive to ill conditioned functional regions, thereby shrinking its basin of attraction.

Data: Boundary Conditions B, Initial damping λ_0
Result:
$\mathbf{p}_0 \longleftarrow$ GetInitialEstimate(B);
$k \longleftarrow 0$;
while *1* **do**
  $J \longleftarrow$ GetJacobian($\mathbf{p}$,B);
  $r \longleftarrow$ GetResidual($\mathbf{p}$,B);
  $\mathbf{g} \longleftarrow$ GetFunctionValues($\mathbf{p}$,B);
  $H \longleftarrow J^\top J$;
  $\nabla r \longleftarrow J^\top \mathbf{g}$;
  while *1* **do**
    $k \longleftarrow k + 1$;
    $\mathbf{d} \longleftarrow (H + \lambda_k I_2)^{-1} \nabla r$;
    if CheckCompletionConditions(k,∇r,$\hat{r}$,λ_k) **then**
      return $\mathbf{p}_k$
    end
    $\hat{\mathbf{p}} \longleftarrow \mathbf{p}_k - \mathbf{d}$;
    $\hat{\mathbf{p}} \longleftarrow$ Project($\hat{\mathbf{p}}$);
    $\hat{r} \longleftarrow$ GetResidual($\hat{\mathbf{p}}$,B);
    if $\hat{r} < r$ **then**
      $\lambda_{k+1} \longleftarrow \lambda_k \alpha$;
      $\mathbf{p}_{k+1} \longleftarrow \hat{\mathbf{p}}$;
      Break Inner While Loop;
    else
      $\lambda_{k+1} \longleftarrow \lambda_k \beta$;
    end
  end
end

Algorithm 6: The Levenberg Marquardt method as implemented for the trajectory generation in this thesis. The values of $\alpha < 1$, $\beta > 1$ governs the rate of change for the damping parameter. The method CheckCompletionConditions can be set to check for various termination criteria, e.g. maximum number of iterations, achieved accuracy, size of the gradient etc. The Project method is described below in Appendix D.2.

D.2 Projection Method

In essence the Projection Method allows unrestricted steps in the chosen search direction, but projects any decision variables ending up outside the feasible set back on it [22]. The procedure is schematically visualized in Figure D.1.

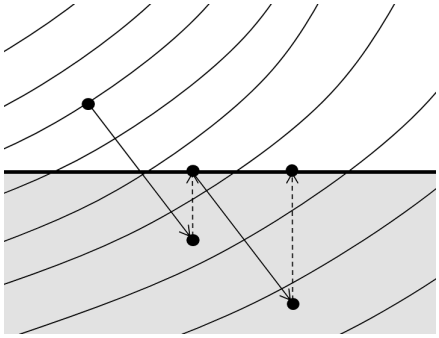


Figure D.1: Schematic illustration of the projection method for handling of inequality constraints. The thicker line represent the boundary of the feasible set (white). The curved lines are contour lines of the objective, solid arrows the search direction at each iterate, dashed arrows the direction of projection

Due to the simple nature of the constraint, the rule is implemented for Equation 3.30 as

$$s_f = \begin{cases} s_f, & \text{if } s_f \geq \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \\ \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}, & \text{otherwise} \end{cases}$$

which is applied on all tentative search directions before they are accepted.

D.3 Scaling of Decision Variables

As noted in both [3] and [21] the system is numerically ill posed as the decision variables commonly differs several orders of magnitude in size. For example, while the final length s_f commonly is in the order of $10^1 - 10^2$, the parameter d (or the equivalent in the work done by others) typically ranges between $10^{-7} - 10^{-6}$. Due to this, numerical stability and faster convergence is gained if the variables are rescaled [22]

Integrating Equation 3.26 one gets

$$\varphi(s) = as + \frac{1}{2}bs^2 + \frac{1}{3}cs^3 + \frac{1}{4}ds^4 \tag{D.5}$$

A scaled equivalent can then be defined as

$$\varphi_s(s_s) = a_s s_s + \frac{1}{2}b_s s_s^2 + \frac{1}{3}c_s s_s^3 + \frac{1}{4}d_s s_s^4 \tag{D.6}$$

where the subscript s differentiating the scaled counterparts of the polynomial coefficients and arclength. If done appropriately the problem as a whole can then be scaled by simultaneously transforming the spatial boundary conditions so that

$$g_{1s}(d_s,s_f) = \int_0^{s_{fs}} \cos(\varphi_s(d_s,s_{fs},s_s))ds_s - x_{1s} = 0 \tag{D.7}$$

$$g_{2s}(d_s,s_f) = \int_0^{s_{fs}} \sin(\varphi_s(d_s,s_{fs},s_s))ds_s - y_{1s} = 0 \tag{D.8}$$

All being measures of distance, the spatial boundary conditions x_1,y_1 and arclength parameter s can for some number N be rescaled as

$$s_s = \frac{s}{N}, \quad x_{1s} = \frac{x_1}{N}, \quad y_{1s} = \frac{y_1}{N}, \quad \kappa_{0s} = \kappa_0 N, \quad \kappa_1 N \tag{D.9}$$

If the scaling of the polynomial coefficients then is done according to

$$a_s = aN \quad b_s = bN^2 \quad c_s = cN^3 \quad d_s = dN^4 \tag{D.10}$$

it is easy to see that

$$\begin{aligned}
\varphi_s(s_s) = \varphi_s\left(\frac{s}{N}\right) &= a_s\frac{s}{N} + \frac{1}{2}b_s\frac{s^2}{N^2} + \frac{1}{3}c_s\frac{s^3}{N^3} + \frac{1}{4}d_s\frac{s^4}{N^4} \\
&= aN\frac{s}{N} + \frac{1}{2}bN^2\frac{s^2}{N^2} + \frac{1}{3}cN^3\frac{s^3}{N^3} + \frac{1}{4}dN^4\frac{s^4}{N^4} \\
&= as + \frac{1}{2}bs^2 + \frac{1}{3}cs^3 + \frac{1}{4}ds^4 = \varphi(s)
\end{aligned}$$

That is, given scaling according to D.9 and D.10 the solution to the original problem can be found as the inversely transformed solution to D.7 and D.8. In practice, the boundary conditions on the spatial coordinates and curvature are transformed upon entry of Algorithm [Algorithm Reference], all other steps performed equally. When the algorithm finishes, it suffices to re transform according to

$$d_s = \frac{d}{N^4}, \quad s_{fs} = \frac{s_f}{N}, \quad \kappa_{0s} = \kappa_0 N, \quad \kappa_1 N \tag{D.11}$$

since the relationship defined in Equation 3.27 holds for the scaled variables, and there-

fore

$$\begin{aligned}
a_s &= \kappa_{0s} = \kappa_0 N = aN \\
b_s &= \left(\frac{12\varphi_1 - 4(2\kappa_{0s} + \kappa_{1s})s_{fs} + d_s s_{fs}^4}{2s_{fs}^2} \right) \\
&= \left(\frac{12\varphi_1 - 4(2\kappa_0 N + \kappa_1 N)\frac{s_f}{N} + dN^4 \frac{s_f^4}{N^4}}{2s_f^2} \right) N^2 \\
&= \left(\frac{12\varphi_1 - 4(2\kappa_0 + \kappa_1)s_f + ds_f^4}{2s_f^2} \right) N^2 = bN^2 \\
c_s &= -3 \left(\frac{4\varphi_1 - 2(\kappa_{0s} + \kappa_{1s})s_{fs} + d_s s_{fs}^4}{2s_{fs}^3} \right) \\
&= -3 \left(\frac{4\varphi_1 - 2(\kappa_0 N + \kappa_1 N)\frac{s_f}{N} + dN^4 \frac{s_f^4}{N^4}}{2s_f^3} \right) N^3 \\
&= -3 \left(\frac{4\varphi_1 - 2(\kappa_0 + \kappa_1)s_f + ds_f^4}{2s_f^3} \right) N^3 = cN^3,
\end{aligned}$$

thereby achieving the desired overall transformation. Even though any number N can be chosen as a basis of this transformation, a convenient choice is the euclidean distance between the spatial end points. In doing this choice, one always puts the target on the unit circle, thereby guaranteeing that the scaled final length must be such that $s_{fs} \geq 1$. This is of practical use in determining if an encountered minima is the desired one, and is used for this purpose in Algorithm 1 presented in the next section. In total, the transformation brings s_f (or really s_{fs}) to lie in the range (1,2) while d_s normally is found within $(-50,50)$.

D.4 Numerical Integration

The integrals containing sines and cosines of the heading polynomial, e.g. 3.25, is of the *Generalized Fresnel* type and lacks analytical solutions due to which numerical methods are needed. Even though a large number of numerical integration techniques can be applied to the problem, *Simpson's Composite Quadrature* rule is chosen due its numerical efficiency when the integrand fullfills certain criteria. The rule is an extension of the more commonly known *Simpson Quadrature*, where the integrand is replaced with a inteprolating Lagrange polynomial of second order, sharing endpoints with the original function. Denoting the original integrand $f(x)$ and the polynomial $p(x)$, the approximate

integral over the range (a,b) is formed as

$$\int_a^b f(x)dx \approx \int_a^b p(x)dx = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right) \tag{D.12}$$

Intended to handle more general classes of functions, the extended version applies the same polynomial approximation over subsections o the integration interval, whereby its accuracy is improved. A schematic representation showing the differences is given in Figure D.2

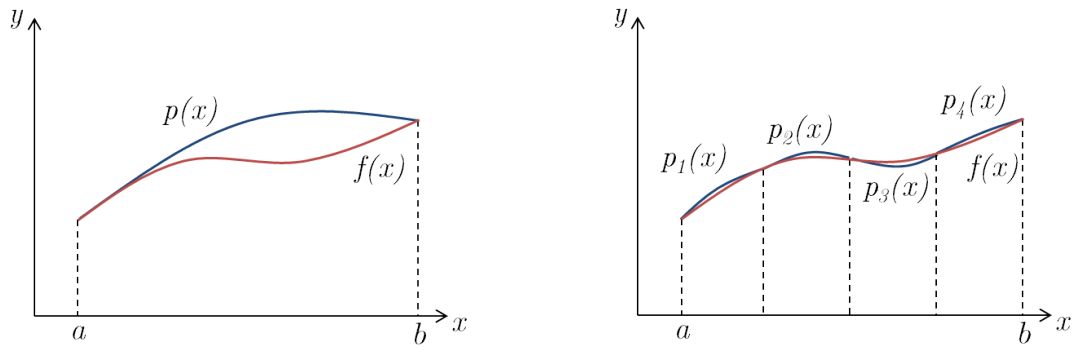


Figure D.2: Illustration of the replacement of the integrand with quadratic polynomials $p(x)$ in Simpsons Quadrature rule (a) and Simpson's Composite Quadrature rule (b). The red line represents the original integrand $f(x)$ and the blue lines the approximating polynomial

The approximation over the range (a,b) are then given as the sum of the polynomial snippets covering each subinterval. It has been shown that the result can be expressed as

$$\int_a^b f(x)dx \approx \frac{\delta}{3}\left(f(a) + 2\sum_{j=1}^{n/2-1} f(a+2j\delta) + 4\sum_{j=1}^{n/2} f(a+\delta(2j-1)) + f(b)\right) \tag{D.13}$$

where n is the number of subintervals. Even though there are modification to this rule for variable segment length, allowing the computational effort spent to be focused on ill behaved, i.e. highly oscillatory or non-smooth, parts of the function $f(x)$, these are avoided for sake of simplicity and the uniform subinterval length rule $\delta = (b-a)/n$ is employed. The error ε in this approximation has been shown to be bounded by

$$|\varepsilon| \leq \frac{(b-a)^5}{180n^4} \max_{\hat{x}\in(a,b)} \left(\left.\frac{d^4 f}{dx^4}\right|_{x=\hat{x}}\right) \tag{D.14}$$

That is, the error decreases with for smoother functions, increased number of intervals n and smaller integration ranges. Given a sufficiently smooth function, the number

of integrand evaluations needed can therefore be kept very low. For details, see any textbook on numerical methods in Calculus, e.g. [58].

The actual expression for the necessary integrals are not given here but are easily derived using standard techniques.

E

Vehicle Modelling

The dynamical vehicle model of choice in this work is a single-track bicycle model displayed in figure E.1. This section aims to mathematically model the lateral transient dynamics of the vehicle. In the following equations, (X,Y) and (x,y) denote the longitudinal and lateral positions in the inertial (global) and the vehicle-fixed (moving) coordinate systems respectively. Subscripts w and v denote the wheel-fixed and the vehicle-fixed axis while front and rear wheels are represented by subscripts f and r.

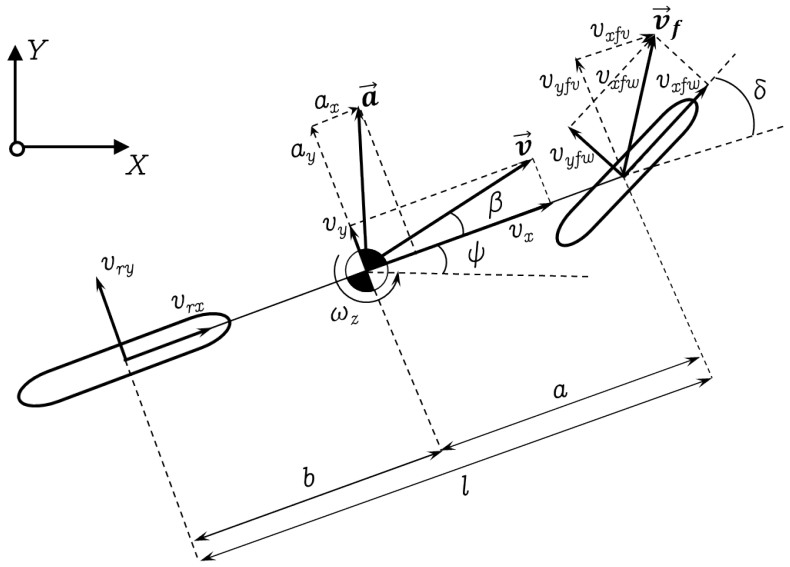
The following holds for the motion of the bicycle in the inertial frame:

$$\begin{aligned} \dot{X} &= v_x \cos\psi - v_y \sin\psi \\ \dot{Y} &= v_x \sin\psi + v_y \cos\psi \\ \dot{\psi} &= \omega_z \end{aligned} \tag{E.1}$$

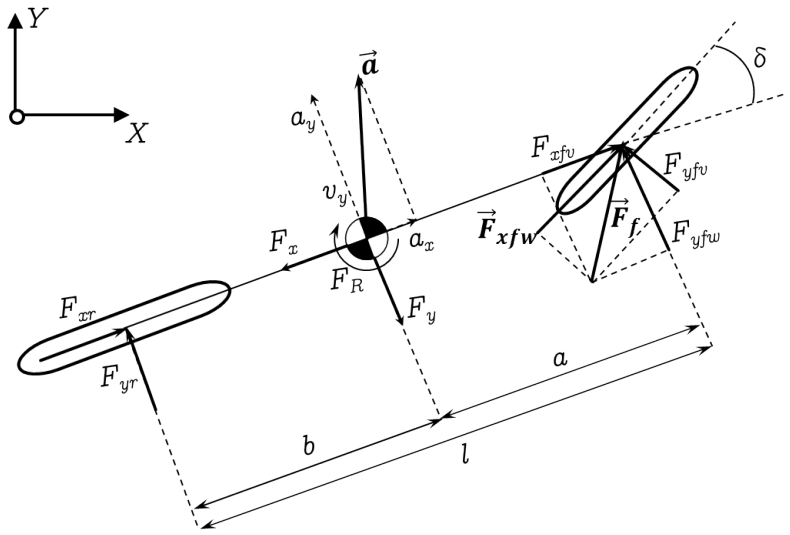
Where ψ represents the yaw angle and ω_z the yaw rate. Assuming linear cornering behaviour yields the following equations for the lateral forces acting on the front and rear wheels:

$$\begin{aligned} F_{yr} &= -C_r \tfrac{v_{yr}}{v_{xr}} \\ F_{yfw} &= -C_f \tfrac{v_{yfw}}{v_{xfw}} \end{aligned} \tag{E.2}$$

Where C_f and C_r represent the cornering stiffness of the front and rear wheels respectively. In the vehicle-fixed frame, the velocities v_{yr}, v_{xr}, v_{yfw} and v_{xfw} are found to



(a) velocities and accelerations



(b) forces

Figure E.1: Accelerations and velocities E.1(a) and forces E.1(b) in the single-track model

be:

$$\begin{aligned} v_{yr} &= v_y - b\omega_z \\ v_{xr} &= v_x \\ v_{yfw} &= (v_y + a\omega_z)\cos(\delta) - v_x \sin(\delta) \\ v_{xfw} &= (v_y + a\omega_z)\sin(\delta) - v_x \cos(\delta) \end{aligned} \tag{E.3}$$

With δ being the steering angle of the front wheel and a and b the distance between the center of gravity and the front and rear axles respectively.

With the knowledge that $a_y = \dot{v}_y + \omega_z v_x$, using Newton's second law, the following dynamics can be derived:

$$\begin{aligned} \dot{v}_x &= \tfrac{1}{m}[F_{xfw}\cos(\delta) - F_{yfw}\sin(\delta) + F_{xr}] + v_y\omega_z \\ \dot{v}_y &= \tfrac{1}{m}[F_{xfw}\sin(\delta) + F_{yfw}\cos(\delta) + F_{yr}] + v_x\omega_z \\ \dot{\omega}_z &= \tfrac{a}{I_z}[F_{xfw}\sin(\delta) + F_{yfw}\cos(\delta)] + \tfrac{b}{I_z}F_{yr} \end{aligned} \tag{E.4}$$

Where m and I_z respectively represent the vehicle's mass and the moment of inertia around the z axis. The front and rear axle forces are denoted by F_{xfw} and F_{xr} respectively. For the case of a font-wheel-drive vehicle and under normal driving conditions, F_{xfw}, and F_{xr} are recognized as the driving force and the rear axle rolling resistance. These parameters shall be set to opposing values of proper magnitude with regard to the desired propulsion. It can be noted that, under the assumption of $\dot{v}_x = 0$, i.e. constant longitudinal velocity, v_x in E.4 will turn into a parameter and the state vector will reduce to $\begin{bmatrix} v_y & \omega_z \end{bmatrix}^T$.

By assuming constant v_x, and small side and tyre slip angles, the model shown in figure E.4 can be simplified to a fully linear model visually represented in figure E.2. For this

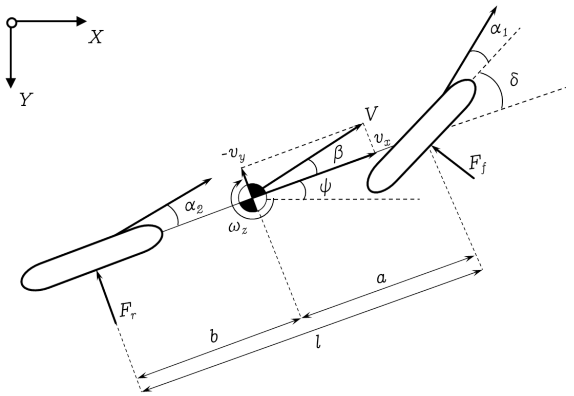


Figure E.2: simplified single-track vehicle model

model the following holds:

$$\begin{aligned} m(\dot{v}_y + v_x\omega_z) &= F_r + F_f \\ I\dot{\omega}_z &= aF_f - bF_r \end{aligned} \tag{E.5}$$

Moreover, the tyre slip angles α_1 and α_2 can be formulated as:

$$\begin{aligned} \alpha_1 &= \delta - \tfrac{1}{v_x}(v_y - b\omega_z) \\ \alpha_2 &= -\tfrac{1}{v_x}(v_y - b\omega_z) \end{aligned} \tag{E.6}$$

Considering linear cornering behaviour yields the following equations for the lateral tyre forces:

$$\begin{aligned} F_f &= C_f\alpha_1 \\ F_r &= C_r\alpha_2 \end{aligned} \tag{E.7}$$

Now, replacing E.7 and E.6 in E.5 gives:

$$\begin{aligned} m\dot{v}_y &= -\tfrac{v_y}{v_x}(C_f + C_r) - mv_x - \tfrac{\omega_z}{v_x}(aC_f - bC_r) + C_f\delta \\ I_z\dot{\omega}_z &= -\tfrac{\omega_z}{v_x}(a^2C_f + b^2C_r) - \tfrac{v_y}{v_x}(aC_f + bC_r) + aC_f\delta \end{aligned} \tag{E.8}$$

In state-space form, the linear model can be presented as:

$$\begin{bmatrix} \dot{v}_y \\ \\ \dot{\omega}_z \end{bmatrix} = \begin{bmatrix} -\frac{C_f+C_r}{mv_x} & -v_x - \frac{aC_f-bC_r}{mv_x} \\ \\ -\frac{aC_f-bC_r}{Iv_x} & -\frac{a^2C_f+b^2C_r}{Iv_x} \end{bmatrix} \begin{bmatrix} v_y \\ \\ \omega_z \end{bmatrix} + \begin{bmatrix} \frac{C_f}{m} \\ \\ \frac{aC_f}{I_z} \end{bmatrix} \delta \tag{E.9}$$

The above model is linear and has the longitudinal speed v_x as a parameter which is assumed to be constant.

As it will be seen further on, the lateral dynamics of the vehicle are of particular interest. Specifically, the lateral acceleration and its variations will be used as criteria for validation of some proposed solutions. It is important to distinguish between the two lateral accelerations $\dot{v}_y$ and a_y. The former is the derivative of the variable v_y in the vehicle-fixed coordinate system while the latter is the lateral acceleration in the inertial coordinate system and can be calculated as $a_y = \dot{v}_y + v_x\omega_z$. In other words, a_y comprises contributions from both the centripetal acceleration $v_x\omega_z$ and the time derivative of v_y. a_y can hence be considered as an output of the models E.4 and E.9. In the remainder of this work, the term *lateral acceleration* of the vehicle refers to a_y as described above.