TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

Bachelor's Thesis in Informatics

Realistic Optimization-based Driving Using a Constrained Double-Integrator Model

Andreas Belavic

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

Bachelor's Thesis in Informatics

Realistic Optimization-based Driving Using a Constrained Double-Integrator Model

Realistisches, optimierungsbasiertes Fahren mit einem beschränkten Doppel-Integrator-Modell

Author:	Andreas Belavic
Supervisor:	Prof. Dr.-Ing. Matthias Althoff
Advisor:	M.Sc. Tobias Mascetta, M.Sc. Lukas Schäfer
Submission Date:	28.02.2025

Abstract

Motion planning is an essential aspect of robotics and autonomous systems, requiring algorithms that efficiently generate safe and feasible trajectories. This thesis examines motion planning for autonomous vehicles by formulating it as an optimization problem that considers safety, efficiency, and computational feasibility. A primary focus is on real-time planning in dynamic environments, where vehicles must navigate while accounting for constraints such as road boundaries, vehicle dynamics, and interactions with other agents.

To address this, we develop optimization-based motion planning approaches using both the point mass and bicycle models. The methodology transforms the motion planning problem into a convex optimization framework, facilitating the efficient computation of feasible trajectories. Techniques such as convex relaxation and quantifier elimination are employed to improve solution robustness and computational efficiency. The proposed approach is evaluated through simulations to assess its performance.

The results indicate that the framework can generate feasible and efficient trajectories while maintaining computational efficiency. The evaluation demonstrates its applicability across various planning scenarios, supporting its potential use in autonomous driving applications.

Contents

Abstract **iii**

1 Introduction **1**

1.1 Overview . . . 1
1.2 Problem Formulation . . . 2
1.3 Related Work . . . 3
1.3.1 Classical Path Planning Approaches . . . 3
1.3.2 Optimization-Based Trajectory Planning . . . 3
1.3.3 Hierarchical and Multi-Layered Architectures . . . 4
1.3.4 Interaction-Aware Planning and Social Compliance . . . 4
1.3.5 Learning-Based Approaches . . . 4
1.3.6 Summary . . . 4
1.4 Thesis Structure . . . 5

2 Preliminaries **6**

2.1 Convex Discrete-Time Optimization . . . 6
2.1.1 Computational Complexity . . . 6
2.1.2 Discrete-Time Problem Formulation . . . 6
2.2 Vehicle Models . . . 7
2.2.1 Point Mass Model . . . 8
2.2.2 Bicycle Model . . . 9
2.2.3 Curve Following Coordinate System . . . 10

3 Methodology **11**

3.1 Motion Planning Using Point Mass . . . 11
3.1.1 Conversion of Global Cartesian Coordinates to Frenet Frame Coordinates . . . 12
3.1.2 System Linearization . . . 13
3.1.3 Eliminating the For-All Operator . . . 17
3.1.4 Determining the Steering Angle . . . 24
3.1.5 Final Model . . . 24
3.2 Motion Planning using Bicycle Model . . . 25
3.2.1 Transforming Global Cartesian Coordinates to Frenet Frame . . . 25

3.2.2 Model Dynamics Approximation . . . 26
3.2.3 Convex Relaxation of Bilinear Terms . . . 28
3.2.4 Final Model . . . 30
3.3 Modeling Techniques . . . 34
3.3.1 Soft Constraints . . . 35
3.3.2 Auxiliary Variables . . . 35
3.3.3 Penalty Methods . . . 36
3.3.4 Conclusion . . . 37

4 Evaluation **38**
4.1 Implementation Details . . . 38
4.1.1 Road Segments . . . 38
4.1.2 Planner . . . 39
4.2 Performance Evaluation . . . 39
4.2.1 Objectives . . . 39
4.2.2 Scenarios . . . 40
4.2.3 Simulation Setup . . . 42
4.2.4 Results . . . 44
4.3 Challenges and Limitations . . . 46
4.3.1 Replanning Challenges for Point Mass . . . 46
4.3.2 Lag in Dynamics . . . 47
4.3.3 McCormick Relaxation . . . 47

5 Discussion and Future Work **50**

6 Conclusion **51**

List of Figures **52**

List of Tables **53**

Bibliography **54**

1 Introduction

1.1 Overview

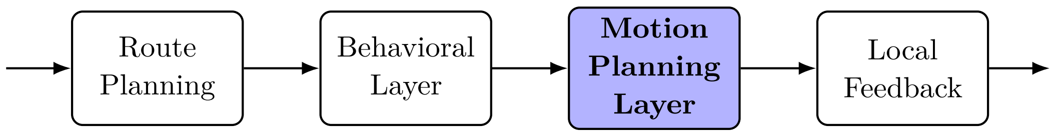


Figure 1.1: Overview of Autonomous Driving Problem Decomposition

The problem of autonomous driving can be divided into four main components, each representing an essential aspect of the system. Figure 1.1 illustrates this decomposition, highlighting the motion planning layer as the focus of this work.

The process begins with the user providing a travel destination, which serves as the input to the Route Planning component. This phase generates a sequence of waypoints through a predefined road network. Next, the Behavioral Layer refines the waypoints by considering environmental factors such as other vehicles, obstacles, and road signs, ensuring the vehicle adapts to dynamic traffic conditions. Once a behavioral strategy is determined, the Motion Planning Layer generates a trajectory that satisfies physical and safety constraints, ensuring feasibility and compliance with road rules. Finally, the Local Feedback component executes the plan by generating precise control commands—steering, throttle, and brake inputs—based on real-time vehicle and environmental feedback.

Finding an exact solution to the motion planning problem is computationally intractable in most cases. As a result, numerical methods are commonly used to approximate solutions. These approaches fall into three main categories:

1. Graph-Based Algorithms discretize the vehicle's state space and connect valid transitions with edges, allowing a graph search algorithm to determine an optimal trajectory

2. Incremental Tree Approaches expand a search tree by randomly applying control commands and simulating state transitions until a feasible trajectory is found.

3. Optimization-Based Methods formulate the problem as an optimization task over a

function space, minimizing an objective function (e.g., travel time, energy efficiency) while respecting constraints.

We focus on optimization-based motion planning, which offers a structured and efficient approach to trajectory generation. Our objective is to design a motion planner that consistently provides near real-time solutions. To achieve this, we leverage modern optimization solvers.

1.2 Problem Formulation

The goal of motion planning is to compute a feasible and optimal trajectory for an autonomous vehicle that safely navigates from an initial state to a goal region while satisfying dynamic and environmental constraints. Using an optimization-based approach, this problem can be formulated as finding a function $\pi(t) : [0, T] \rightarrow \mathcal{X}$,where $\mathcal{X}$ is the configuration space representing all feasible vehicle states, and T is the planning horizon.

The vehicle starts at an initial configuration $x_{\text{initial}} \in \mathcal{X}$, and must reach a goal region $X_{\text{goal}} \subset \mathcal{X}$ within the time horizon T . The trajectory must also satisfy constraints on its derivatives, such as velocity, acceleration, and higher-order dynamics, which are represented by a predicate $D(\pi(t), \pi'(t), \pi''(t), \dots)$.

To formulate this as an optimization problem, we define an objective function $J(\pi) : \Pi[\mathcal{X}, T] \rightarrow \mathbb{R}$ that evaluates the quality of a given trajectory $\pi(t)$ over the planning horizon. This function may incorporate metrics such as smoothness, energy efficiency, safety, or time minimization.

Problem Definition: Optimal Trajectory Planning

Given a 6-tuple $(\mathcal{X}, x_{\text{initial}}, X_{\text{goal}}, D, J, T)$, the task is to find:

$$
\begin{aligned}
x^* = \; & \arg\min_{\pi \in \Pi(\mathcal{X}, T)} J(\pi) && && (1.1) \\
\text{s.t.} \quad & \pi(0) = x_{\text{initial}} && && (1.2) \\
& \pi(T) \in X_{\text{goal}} && && (1.3) \\
& \pi(t) \in \mathcal{X}, && \text{for all} \quad t \in [0, T] && (1.4) \\
& D(\pi(t), \pi'(t), \pi''(t), \dots), && \text{for all} \quad t \in [0, T] && (1.5)
\end{aligned}
$$

This formulation ensures that the computed trajectory is feasible, adheres to the vehicle's dynamic constraints, and optimizes a desired cost function. The next sections will discuss methods for solving this problem efficiently using numerical optimization techniques.

1.3 Related Work

Since the early demonstrations in the DARPA Grand and Urban Challenges [1][2], the motion planning literature for self-driving vehicles has evolved to address a broad range of environments—from structured highways and city roads to unstructured parking lots and off-road terrains. A unifying theme across these scenarios is the need to produce collision-free, dynamically-feasible trajectories that also account for comfort and operational constraints.

1.3.1 Classical Path Planning Approaches

Initial research in autonomous driving often used graph-based or search-based algorithms, which discretize the state space into grids or motion “primitives” and then apply search algorithms such as A* or D* [3]. While these methods can systematically find a solution if one exists, they can become computationally expensive in higher-dimensional state spaces, especially when including vehicle dynamics. Another popular family of methods is sampling-based planners, including Rapidly-exploring Random Trees (RRT) and Probabilistic Roadmaps (PRM) [4]. For on-road driving, variants like RRT* incorporate cost optimization and can quickly explore feasible regions. However, sampling alone does not guarantee smoothness or dynamic feasibility, often requiring a secondary smoothing or optimization step.

1.3.2 Optimization-Based Trajectory Planning

In contrast to sampling-based methods, optimization-based trajectory planning explicitly encodes the cost criteria and constraints—such as collision avoidance, road boundary constraints, and vehicle dynamics—into an objective function. A trajectory is then optimized continuously rather than constructed through random sampling. One prominent approach employs Frenet frames to transform the planning problem from a Cartesian to a curvilinear coordinate system, simplifying the computation of collision and road boundary constraints [5]. This representation has proven effective for lane-keeping, lane changes, and freeway driving, where the environment can be reasonably approximated by a reference path.

A closely related technique is Model Predictive Control (MPC) [6][7], which casts trajectory planning as a finite-horizon optimal control problem, repeatedly solved online. By incorporating vehicle dynamics, actuator limits, and predicted movements of surrounding vehicles, MPC can adapt in real time to changing traffic conditions. Variants such as Nonlinear MPC (NMPC) can handle more complex or higher-fidelity vehicle models (e.g., for aggressive maneuvers), but typically require significant computational resources. Recent work has explored stochastic MPC formulations to account for measurement and prediction uncertainties [8], aiming to provide robust performance under noisy sensor data and uncertain driver intentions.

1.3.3 Hierarchical and Multi-Layered Architectures

Many autonomous driving systems employ a hierarchical architecture that separates global route planning from local trajectory optimization. At the global level, simpler search-based or graph-based methods can select a route through the road network (e.g., from a start location to a destination). At the local level, an optimization-based planner refines this route into a dynamically-feasible, collision-free trajectory in real time [9]. This layered approach leverages the strengths of each method: global planners handle large-scale navigation, while local planners ensure feasibility and handle dynamic obstacles, road curvature, and driving rules.

1.3.4 Interaction-Aware Planning and Social Compliance

A significant challenge in urban environments is predicting how other road users—vehicles, cyclists, and pedestrians—will move and interact. Behavior prediction and interaction-aware planning thus play a critical role in modern autonomous driving pipelines [10]. One line of research integrates game-theoretic or multiagent models into the trajectory planning stage, enabling the autonomous vehicle to reason about how human drivers might respond. Alternatively, rules can be added to make the vehicle behave more cautiously around pedestrians or follow right-of-way rules at intersections.

1.3.5 Learning-Based Approaches

Recent advances in machine learning have led to the development of learning-based trajectory planning systems. These can take the form of imitation learning—where the planner learns a cost function or policy from human driving data [11]—or deep reinforcement learning, where the agent optimizes a reward function through simulation [12]. Nevertheless, pure data-driven methods can lack the stability and formal guarantees of classical optimization. As a result, hybrid methods that use learning to shape the objective or constraints, while retaining an optimization-based trajectory generator for low-level feasibility, have seen increased popularity [13].

1.3.6 Summary

In summary, the field of motion planning for autonomous driving has seen significant advancements over the years, evolving from classical path planning approaches to more sophisticated optimization-based and learning-based methods. Each approach has its strengths and weaknesses, and the choice of method often depends on the specific driving scenario and requirements. Classical methods like graph-based and sampling-based planners provide systematic solutions but can struggle with high-dimensional state spaces and dynamic feasibility. Optimization-based methods offer a more continuous and constraint-aware approach, particularly effective in structured environments. Hierarchical architectures combine global and local planning to handle large-scale navigation and real-time trajectory optimization. Interaction-aware planning addresses the complexities of urban environments by predicting and responding to the behavior of other road users. Finally, learning-based approaches leverage data-driven techniques to improve planning performance, though they often benefit from hybridization with classical methods to ensure stability and feasibility. As research continues, the integration of these diverse techniques promises to enhance the safety, efficiency, and robustness of autonomous driving systems.

1.4 Thesis Structure

This thesis is organized into six chapters. Chapter 2 introduces the fundamental concepts necessary for the methodology, including convex discrete-time optimization and vehicle modeling approaches such as the point mass and bicycle models. Chapter 3 presents the proposed motion planning methods, detailing coordinate transformations, constraint formulations, model dynamics approximations, and convex relaxation techniques. It also explores various modeling strategies, including soft constraints and auxiliary variables. Chapter 4 focuses on the implementation and evaluation of the proposed methods, describing performance analysis, simulation results, and challenges. Chapter 5 discusses the findings, their implications, and possible improvements, outlining future research directions. Finally, Chapter 6 concludes the thesis by summarizing its key contributions and highlighting the broader impact of the research.

2 Preliminaries

2.1 Convex Discrete-Time Optimization

2.1.1 Computational Complexity

Finding an exact solution to the optimal trajectory planning problem, as formulated in Section 1.2, is generally computationally intractable. Specifically, this problem belongs to the class of PSPACE-Hard problems, meaning that the required computational resources grow exponentially with the problem size. The complexity arises from several factors, including the continuous nature of the state space, the non-convex constraints imposed by vehicle dynamics, and the need to account for obstacles and environmental uncertainties over a planning horizon. This challenge is closely related to the classic Movers' Problem [14], which is PSPACE- Hard and involves finding a collision-free path for a robot in an environment with obstacles. Motion planning for autonomous vehicles introduces additional complexities, such as dynamic constraints and time dependencies, further increasing the computational burden. Given these challenges, exact solutions are impractical for real-time applications. Instead, numerical optimization techniques, heuristics, and approximate solvers are employed to compute near-optimal solutions efficiently. The following sections describe the methods used in this work to address these challenges while ensuring computational feasibility.

2.1.2 Discrete-Time Problem Formulation

To make the trajectory planning problem more tractable, we discretize the continuous-time problem into a finite set of time steps. Let $\mathcal{X}$ denote the set of valid vehicle states and $\mathcal{U}$ the set of feasible control inputs.

The trajectory is defined at discrete time points $\{t_i\}_{i=1,\dots,m}$, where $\pi(t_i) = x_i$. The objective function is then formulated over $\mathcal{X} \times \mathcal{U}$ as $J : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$.

Problem Definition: Discrete-Time Optimal Trajectory Planning

Given a 7-tuple $(\mathcal{X}, \mathcal{U}, x_{\text{initial}}, X_{\text{goal}}, f, J, \{t_i\}_{i=1,\dots,m})$, find:

$$u^* = \arg\min_{u \in \mathcal{U}^{T-1}} \sum_{i=1}^{T-1} J(x_{i+1}, u_i) \tag{2.1}$$

$$\text{s.t.} \quad x_1 = x_{\text{initial}} \tag{2.2}$$

$$x_T \in X_{\text{goal}} \subseteq \mathcal{X} \tag{2.3}$$

$$(x_i, u_i) \in \mathcal{C} \subseteq \mathcal{X} \times \mathcal{U} \qquad \text{for all } i \in \{1, \ldots, m-1\} \tag{2.4}$$

$$x_{i+1} = x_i + (t_{i+1} - t_i) f(x_i, u_i) \qquad \text{for all } i \in \{1, \ldots, m-1\} \tag{2.5}$$

This formulation introduces a coupling constraint C that imposes restrictions on both the state and control inputs. The dynamics are approximated using a first-order Euler integration scheme, where the state at time $t_i + 1$ is computed based on the state at time t_i and the control input u_i.

Disciplined Convex Programming (DCP)

The DCP framework imposes specific rules on how optimization problems must be formulated, which helps in verifying the convexity of the problem and guarantees that the problem can be solved efficiently. The key principles of DCP are as follows:

- The objective function must be convex if it is to be minimized, or concave if it is to be maximized.
- Constraints must be formulated in one of the following forms:
  - An equality constraint between affine expressions: affine = affine
  - An inequality constraint where a convex function is less than or equal to a concave function: convex $\leq$ concave

The use of DCP in this work involves defining the cost function and constraints in a manner that satisfies the DCP rules. We will use the term "convex constraint" or "convex form" to refer to constraints that adhere to these rules.

2.2 Vehicle Models

A vehicle model defines the position and orientation of a vehicle in the real world and predicts how these states evolve over time. The choice of model significantly impacts the accuracy and computational complexity of motion planning. Simpler models, while computationally efficient, may lack precision, whereas more complex models provide higher fidelity but require greater computational resources.

This section introduces two fundamental models used in trajectory planning [15]: the point mass model, which provides a simplified kinematic representation, and the bicycle model, which captures essential vehicle dynamics. In both models, the system's state is represented by a vector x, containing relevant variables such as position, velocity, and orientation, while the control inputs, represented by a vector u, influence state transitions over time. The vehicle dynamics for both models can be generally expressed as:

$$\dot{x} = f(x, u) \tag{2.6}$$

where $f(x, u)$ represents the system dynamics as a function of the state vector x and the control inputs u.

2.2.1 Point Mass Model

The point mass model (PM) represents the vehicle as a point in space with velocity components along the x- and y-axes. It simplifies vehicle motion by ignoring orientation and steering dynamics, making it computationally efficient for trajectory optimization. The vehicle state is represented by four variables:

$$x = \begin{bmatrix} p_x \\ p_y \\ v_x \\ v_y \end{bmatrix} \tag{2.7}$$

where p_x and p_y define the position in a fixed global coordinate system, while v_x and v_y represent the velocity components in the respective directions.

The control inputs are:

$$u = \begin{bmatrix} a_x \\ a_y \end{bmatrix} \tag{2.8}$$

where a_x and a_y denote accelerations along the x- and y-axes. The system follows the linear dynamics:

$$\dot{x} = \begin{bmatrix} v_x \\ v_y \\ a_x \\ a_y \end{bmatrix} \tag{2.9}$$

Additionally, the control inputs are constrained by:

$$\sqrt{u_1^2 + u_2^2} \leq a_{\max} \tag{2.10}$$

where $a_m ax$ limits the maximum allowable acceleration. Despite its simplicity, the point mass model is widely used for trajectory planning due to its convex formulation and computational efficiency. However, it lacks the ability to represent steering dynamics and vehicle orientation, making it less suitable for applications requiring high-precision maneuvering.

2.2.2 Bicycle Model

The bicycle model, also known as the kinematic single-track model (KST), provides a more detailed representation of vehicle motion by incorporating orientation and steering dynamics. This model is well-suited for trajectory planning in autonomous driving as it captures essential vehicle behavior while remaining computationally tractable.

The state vector consists of five variables:

$$x = \begin{bmatrix} p_x \\ p_y \\ \delta \\ v \\ \psi \end{bmatrix} \tag{2.11}$$

where:

- p_x and p_y represent the vehicle's position in the global coordinate system,
- δ is the front-wheel steering angle,
- v is the longitudinal velocity of the rear wheel, and
- ψ is the vehicle's orientation relative to the global x-axis.

The control inputs are:

$$u = \begin{bmatrix} v_\delta \\ a_x \end{bmatrix} \tag{2.12}$$

where v_δ is the steering velocity, and a_{long} is the longitudinal acceleration.

The vehicle's motion follows the kinematic equations:

$$\dot{p}_x = v\cos(\psi) \tag{2.13}$$

$$\dot{p}_y = v\sin(\psi) \tag{2.14}$$

$$\dot{\delta} = v_\delta \tag{2.15}$$

$$\dot{v} = a_{\text{long}} \tag{2.16}$$

$$\dot{\psi} = \frac{v}{l_{wb}}\tan(\delta) \tag{2.17}$$

$$\tag{2.18}$$

where $l_w b$ represents the wheelbase length.

The single-track name arises from approximating both front and rear wheels as single contact points, assuming no lateral slip. This simplification makes the model suitable for path planning while still capturing fundamental steering dynamics.

The bicycle model is visualized in Figure 2.1.

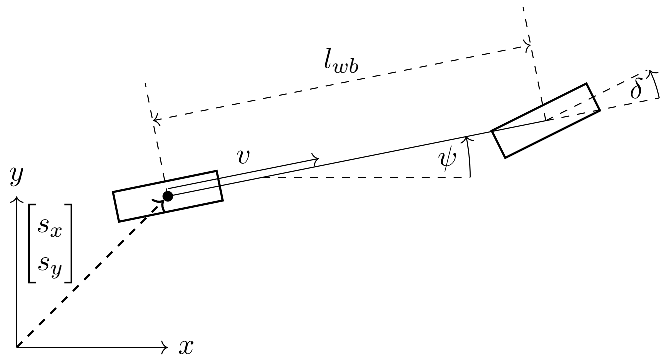


Figure 2.1: Bicycle model representation of a vehicle.

Additional constraints ensure realistic vehicle behavior. A maximum acceleration constraint is imposed:

$$\sqrt{u_2^2 + (x_4 \dot{x}_5)^2} \leq a_{\max} \tag{2.19}$$

For both models, the vehicle is additionally constrained by its velocity range, steering angle range, and the rate of change of the steering angle. These constraints are natural and should not be overlooked.

2.2.3 Curve Following Coordinate System

In addition to the global coordinate system, a curve-following coordinate system, commonly known as the Frenet frame, can be used to simplify the representation of vehicle motion along a predefined path. The Frenet frame consists of the arc-length coordinate s and the lateral offset n from the path, as shown in Figure 2.2.

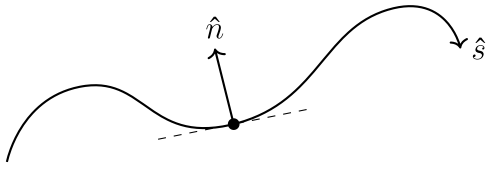


Figure 2.2: Frenet Frame Representation

3 Methodology

3.1 Motion Planning Using Point Mass

In Section 2.2.1, we introduced the point mass model with global coordinates and no orientation. We now extend this model to the Frenet frame, which allows for a more structured approach to motion planning along a predefined road. The idea and the equations are based on the work of Eilbrecht et al. [1].

To achieve this, we first define the curvature of the reference path, which quantifies the rate of change of the tangent angle with respect to arc length:

$$C := \frac{d\theta}{ds}. \tag{3.1}$$

Additionally, let ψ be the orientation of the vehicle, and define the alignment error ξ as the difference between the vehicle's orientation and the reference path's tangent angle:

$$\xi := \psi - \theta. \tag{3.2}$$

This alignment error measures the deviation of the vehicle's heading from the reference path.

Using these definitions, along with established kinematic relationships and coordinate frame transformations [1], we can systematically derive the motion dynamics in the Frenet frame. These equations describe how the vehicle's velocity components in the body-fixed frame relate to changes in Frenet frame coordinates.

The first-order derivatives are given by:

$$\dot{s}(1 - nC(s)) = v_x \cos\xi - v_y \sin\xi, \tag{3.3}$$

$$\dot{n} = v_x \sin\xi + v_y \cos\xi. \tag{3.4}$$

These equations describe the evolution of the vehicle's longitudinal and lateral positions in the Frenet frame. The presence of the curvature term $C(s)$ ensures that the equations correctly account for the curvature of the reference path.

For the second derivatives, we obtain:

$$a_{x,tn} = (a_x - v_y\dot{\psi}) \cos\xi - (a_y + v_x\dot{\psi}) \sin\xi, \tag{3.5}$$

$$a_{y,tn} = (a_x - v_y\dot{\psi}) \sin\xi + (a_y + v_x\dot{\psi}) \cos\xi. \tag{3.6}$$

where the transformed acceleration terms in the Frenet frame are given by:

$$a_{x,tn} := \ddot{s}(1 - nC(s)) - 2\dot{n}C(s)\dot{s} - nC'(s)\dot{s}^2, \tag{3.7}$$

$$a_{y,tn} := \ddot{n} + C(s)\dot{s}^2(1 - nC(s)). \tag{3.8}$$

In subsequent sections, we will leverage these equations to develop motion planning strategies that optimize trajectory feasibility and control performance.

3.1.1 Conversion of Global Cartesian Coordinates to Frenet Frame Coordinates

We now define the state variables x_0, representing the vehicle's position, orientation, and velocities in the Frenet frame, along with the control inputs u_0 for our point mass model in the Frenet frame:

$$x_0 = \begin{bmatrix} s & n & \xi & \dot{s} & \dot{n} & \dot{\psi} \end{bmatrix}^T \tag{3.9}$$

The state vector consists of the Frenet frame coordinates that define the vehicle's location, the alignment error representing the deviation of the vehicle's orientation from the reference path, and the first-order time derivatives of these quantities, which capture the vehicle's velocity components.

We define the following three control inputs:

$$u_0 = \begin{bmatrix} a_x & a_y & a_\psi \end{bmatrix}^T \tag{3.10}$$

Using the previously derived equations, we can formulate the model dynamics in the Frenet frame as:

$$f_0(x_0, u_0) = \begin{bmatrix} \dot{s} \\ \dot{n} \\ \dot{\psi} - C(s)\dot{s} \\ \dfrac{a_{x,tn} + 2\dot{n}C(s)\dot{s} + nC'(s)\dot{s}^2}{1 - nC(s)} \\ a_{y,tn} - C(s)\dot{s}^2(1 - nC(s)) \\ a_\psi \end{bmatrix} \tag{3.11}$$

With this formulation, we are now able to model vehicle motion in the Frenet frame along a predefined road. However, the inclusion of curvature terms introduces non-convexity into the system dynamics. In the following section, we will explore methods to address this challenge and develop techniques to handle the resulting non-convex constraints effectively.

3.1.2 System Linearization

First, we decouple the input. By examining the dynamics equations (3.11) and substituting the equations (3.7) and (3.8), we observe that the last two entries both contain the control inputs a_x and a_y. We proceed by making the following assumption:

Assumption: Alignment Error

Orientation of the vehicle ψ equals the angle of the road θ:

$$\xi = \psi - \theta = 0 \tag{3.12}$$

which directly implies the following three points, which will help us in further steps:

$$[a_x, a_y] = [a_{x,tn}, a_{y,tn}] \tag{3.13}$$

$$\dot{\psi} = \dot{\theta} = \frac{d\theta}{ds} \cdot \frac{ds}{dt} = C(s)\dot{s} \tag{3.14}$$

$$a_\psi = \ddot{\psi} = \ddot{\theta} = C'(s)\dot{s}^2 + C(s)\ddot{s} \tag{3.15}$$

At first glance, it seems like we are removing the orientation from our model, but we actually just force the vehicle to always be aligned with the road. And since we allow lateral acceleration of the vehicle, it is not fixed to a constant offset to the reference path and can still move left or right.

With this assumption we end up with dynamics equation, which are affine linear in $a_{x,tn}$ and $a_{y,tn}$. This allows us to introduce artificial control inputs to linearize the dynamics.

$$\tilde{u} := \begin{bmatrix} u_t \\ u_n \end{bmatrix} := \begin{bmatrix} \dfrac{a_{x,tn} + 2\dot{n}C(s)\dot{s} + nC'(s)\dot{s}^2}{1 - nC(s)} \\ a_{y,tn} - C(s)\dot{s}^2(1 - nC(s)) \end{bmatrix} \tag{3.16}$$

This procedure is called Feedback Linearization and described in depth by [2]. We want to shortly introduce the topic.

Feedback Linearization

Feedback linearization is a nonlinear control technique that transforms a nonlinear system into an equivalent linear system by means of a suitable change of variables and an appropriate state-feedback law. This approach allows one to apply classical linear control design methods to inherently nonlinear systems.

Consider a general nonlinear system of the form:

$$\dot{x} = f(x) + G(x)\,u, \tag{3.17}$$

where

- $x \in \mathbb{R}^n$ is the state vector,
- $f(x)$ represents the system's dynamics,
- $G(x)$ is the input matrix,
- $u \in \mathbb{R}^m$ is the control input vector.

For feedback linearization, we typically assume the system is *fully actuated*, which requires that

$$\operatorname{rank}\Big(G(x)\Big) = n,$$

i.e., $G(x)$ must be invertible for all x in the region of interest. This condition ensures that the control input u directly influences every component of the state vector.

If $G(x)$ is invertible, then we can solve for the control input u in terms of the state x, its dynamics, and a new artificial control input v. Specifically,

$$u = G(x)^{-1}\Big[v - f(x)\Big]. \tag{3.18}$$

By choosing u in this way, the nonlinear dynamics $f(x)$ are effectively canceled, leaving the new input v to be designed (using standard linear control techniques) to achieve the desired closed-loop behavior.

Resulting Simplified Model

Since the orientation is fixed, we can remove it from the state variables, and we end up with the following state variables.

$$x_1 = \Big[s, \quad n, \quad \dot{s}, \quad \dot{n}\Big]^T$$

We use the new defined artificial controls inputs, for controlling the system.

$$\tilde{u} = \Big[u_t, \quad u_n\Big]^T$$

The dynamics are given by:

$$f_1(x_1, \tilde{u}) = \begin{bmatrix} \dot{s} \\ \dot{n} \\ u_t \\ u_n \end{bmatrix} \tag{3.19}$$

Constraints

We have addressed the dynamics constraints of our discrete-time optimal trajectory planning problem (2.5). Next, we will define the coupling constraints on the state variables and control inputs, discuss the arising challenges, and explore potential solutions.

First, let's examine the vehicle constraints [1]. Let $\square$ represent one of the variables used in trajectory planning. The upper bound is denoted by $\overline{\square}$, and the lower bound by $\underline{\square}$, both of which remain constant during planning.

Constraints on the velocity are defined in a body-fixed manner, with upper and lower bounds, using the following annotations.

$$\underline{v_x} \leq v_x \leq \overline{v_x} \tag{3.20}$$
$$\underline{v_y} \leq v_y \leq \overline{v_y} \tag{3.21}$$

We can easily apply (3.4) and (3.3) to get resulting constraints on our state variables. The equations reduce together with $\xi = 0$ to:

$$\underline{v_x} \leq \dot{s}(1 - nC(s)) \leq \overline{v_x} \tag{3.22}$$
$$\underline{v_y} \leq \dot{n} \leq \overline{v_y} \tag{3.23}$$

For the acceleration two types of constraints are usually defined, the first one which constrains the relations of the longitudinal and lateral acceleration as:

$$a_x^2 + a_y^2 \leq c \tag{3.24}$$

for some constant radius $c \in \mathbb{R}^+$, the second similar to the velocity as follows:

$$\underline{a_x} \leq a_x \leq \overline{a_x} \tag{3.25}$$
$$\underline{a_y} \leq a_y \leq \overline{a_y} \tag{3.26}$$

Using our derived equations, we can define a mapping from the model's state variables and artificial control inputs to the body-fixed accelerations.

$$g(x_1, \tilde{u}) := \begin{bmatrix} (1 - nC(s))u_t - (2\dot{n}C(s)\dot{s} + nC'\dot{s}^2) \\ u_n + C(s)\dot{s}^2(1 - nC(s)) \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \end{bmatrix} \tag{3.27}$$

Combining the vehicle constraints derived from the body-fixed frame and the straight-forward constraints given by the road, we can define our coupling constraint set $\mathcal{C}$

as:

$$\mathcal{C} := \left\{ \begin{bmatrix} x_1 \\ \tilde{u} \end{bmatrix} \middle| \begin{array}{l} \underline{s} \leq s \leq \overline{s}, \\ \underline{n}(s) \leq n \leq \overline{n}(s), \\ \underline{v_x} \leq \dot{s}(1 - nC(s)) \leq \overline{v_x} \\ \underline{v_y} \leq \dot{n} \leq \overline{v_y} \\ \underline{\dot{\psi}} \leq C(s)\dot{s} \leq \overline{\dot{\psi}}, \\ \underline{a_\psi} \leq C'\dot{s}^2 + C(s)u_t \leq \overline{a_\psi}, \\ \begin{bmatrix} \underline{a_x} \\ \underline{a_y} \end{bmatrix} \leq g(x_1, \tilde{u}) \leq \begin{bmatrix} \overline{a_x} \\ \overline{a_y} \end{bmatrix} \\ ||g(x_1, \tilde{u})||^2 \leq c \end{array} \right\} \tag{3.28}$$

This set is highly non-convex, and we now face the challenge of transforming these constraints into a convex form. We will address this by finding an inner polytope of the set $\mathcal{C}$ as follows.

Problem Definition: Finding an Inner Polytope

Given set $\mathcal{C}$ over the state variables x_1 and control inputs $\tilde{u}$, find $\underline{\mathcal{C}}$, such that:

$$\underline{\mathcal{C}} = \left\{ \begin{bmatrix} x_1 \\ \tilde{u} \end{bmatrix} \middle| N \begin{bmatrix} x_1 \\ \tilde{u} \end{bmatrix} \leq b \right\} \subseteq \mathcal{C} \tag{3.29}$$

We will demonstrate two methods to achieve this goal, both aiming to accomplish the following:

We want to find the set $\underline{\tilde{\mathcal{C}}}$ over the variables $\dot{s}$ and $\tilde{u}$, such that:

$$\underline{\tilde{\mathcal{C}}} = \left\{ \begin{bmatrix} \dot{s} \\ u_t \\ u_n \end{bmatrix} \middle| \begin{bmatrix} x_1 \\ \tilde{u} \end{bmatrix} \in \mathcal{C}, \text{for all } \begin{bmatrix} s \\ n \\ \dot{n} \end{bmatrix} \in \begin{bmatrix} \underline{s}, \overline{s} \\ \underline{n}, \overline{n} \\ \underline{\dot{n}}, \overline{\dot{n}} \end{bmatrix} \right\} \tag{3.30}$$

Specifically, we aim to find a set of linear inequalities, each representing a half-space, such that the intersection of these half-spaces forms the set $\underline{\tilde{\mathcal{C}}}$. The following condition should then be satisfied:

$$\begin{bmatrix} \dot{s} \\ u_t \\ u_n \end{bmatrix} \in \underline{\tilde{\mathcal{C}}} \implies \forall \begin{bmatrix} s \\ n \\ \dot{n} \end{bmatrix} \in \begin{bmatrix} \underline{s}, \overline{s} \\ \underline{n}, \overline{n} \\ \underline{\dot{n}}, \overline{\dot{n}} \end{bmatrix} : \begin{array}{ll} (\underline{v_x} \leq \dot{s}(1 - nC(s)) \leq \overline{v_x} & \wedge \\ \underline{\dot{\psi}} \leq C(s)\dot{s} \leq \overline{\dot{\psi}} & \wedge \\ \underline{a_\psi} \leq C'\dot{s}^2 + C(s)u_t \leq \overline{a_\psi} & \wedge \\ \begin{bmatrix} \underline{a_x} \\ \underline{a_y} \end{bmatrix} \leq g(x_1, \tilde{u}) \leq \begin{bmatrix} \overline{a_x} \\ \overline{a_y} \end{bmatrix} & \wedge \\ ||g(x_1, \tilde{u})||^2 \leq c) & \end{array} \tag{3.31}$$

3.1.3 Eliminating the For-All Operator

Solving the problem of finding an inner polytope of the set $\mathcal{C}$ requires addressing the universal quantifier in (3.31). We will explore two methods to eliminate this operator and find a polytope that satisfies the constraints for all possible values of the state variables s, n, and $\dot{n}$.

Interval Fitting for State Variables and Control Inputs

The first approach is computationally less expensive but results in a smaller feasible set. Despite this, it performs remarkably well. The idea is to find intervals for the variables of interest, namely $\dot{s}$, u_t, and u_n, such that for any values these variables may take within those intervals, the implications from (3.31) hold. Each condition of these implications follows a specific pattern. Let $x \in \{\dot{s}, u_t, u_n\}$ and y be a vector containing the remaining variables that are part of the condition.

$$c_{min} \leq f(x, y) \leq c_{max} \tag{3.32}$$

where $x \in \mathbb{R}$, $y \in \mathbb{R}^n$, and $f : \mathbb{R}^{n+1} \to \mathbb{R}$, with constants $c_{min}, c_{max} \in \mathbb{R}$. The variable x is selected such that all other variables contained in y are bounded. Additionally, x is chosen so that f is affine in x, represented by:

$$f(x, y) = a(y)x + b(y) \tag{3.33}$$

where $a, b : \mathbb{R}^n \to \mathbb{R}$. Since a and b are continuous functions over a bounded domain Y, we can find bounds on $a(y)$ and $b(y)$:

$$a_{min} \leq a(y) \leq a_{max}, \quad b_{min} \leq b(y) \leq b_{max} \tag{3.34}$$

Our goal is to find an interval $[\underline{x}, \overline{x}]$ for x such that

$$x \in [\underline{x}, \overline{x}] \implies \forall y \in Y : c_{min} \leq f(x, y) \leq c_{max} \tag{3.35}$$

We define $X := [\underline{x}, \overline{x}]$. To calculate X, we perform a case distinction based on the possible signs of $a(y)$. Let's start with

1. $a(y) > 0$: We can subtract (3.32) with $b(y)$ and divide by $a(y)$:

$$\frac{c_{min} - b(y)}{a(y)} \leq x \leq \frac{c_{max} - b(y)}{a(y)}$$

Since we have to ensure the condition holds even in the worst case, X is given by:

$$\underline{x} = \begin{cases} \dfrac{c_{min} - b_{min}}{a_{max}}, & \text{if } c_{min} - b_{min} < 0 \\ \dfrac{c_{min} - b_{min}}{a_{min}}, & \text{otherwise} \end{cases}$$

$$\overline{x} = \begin{cases} \dfrac{c_{max} - b_{max}}{a_{min}}, & \text{if } c_{max} - b_{max} < 0 \\ \dfrac{c_{max} - b_{max}}{a_{max}}, & \text{otherwise} \end{cases}$$

2. $a(y) \geq 0$: Since $a(y) = 0$ for some $y \in Y$, we have to test if this condition holds:

$$c_{min} \leq b_{min} \text{ and } b_{max} \leq c_{max}$$

If this condition does not hold, then $X = \emptyset$. Otherwise, we exclude all $y \in Y$ for which $a(y) = 0$ and proceed to the first case.

3. $a(y) < 0$: We can again subtract $b(y)$ from (3.32) and divide by $a(y)$, but this time the direction of the inequalities changes:

$$\frac{c_{max} - b(y)}{a(y)} \leq x \leq \frac{c_{min} - b(y)}{a(y)}$$

and by looking at the worst cases of the lower and upper bound, we can calculate X:

$$\underline{x} = \begin{cases} \dfrac{c_{max} - b_{max}}{a_{max}}, & \text{if } c_{max} - b_{max} < 0 \\ \dfrac{c_{max} - b_{max}}{a_{min}}, & \text{otherwise} \end{cases}$$

$$\overline{x} = \begin{cases} \dfrac{c_{min} - b_{min}}{a_{min}}, & \text{if } c_{min} - b_{min} < 0 \\ \dfrac{c_{min} - b_{min}}{a_{max}}, & \text{otherwise} \end{cases}$$

4. $a(y) \leq 0$: Similar to the second case, we need to check if $c_{min} \leq b_{min}$ and $b_{max} \leq c_{max}$ hold. If not, set $X = \emptyset$. Otherwise, ignore the values where $a(y)$ equals zero and proceed to the third case.

5. We have so far considered all the cases where $a(y)$ cannot take both positive and negative values. We now consider the last case, where $a_{min} < 0$ and $a_{max} > 0$. Since $a(y) = 0$ for some values, we first check if $c_{min} \leq b_{min}$ and $b_{max} \leq c_{max}$. If this condition does not hold, we set $X = \emptyset$. Otherwise, X is given by:

$$\underline{x} = \max\left\{\frac{c_{min} - b_{min}}{a_{max}}, \frac{c_{max} - b_{max}}{a_{min}}\right\}$$

$$\overline{x} = \min\left\{\frac{c_{max} - b_{max}}{a_{max}}, \frac{c_{min} - b_{min}}{a_{min}}\right\}$$

If we end up with $x_{max} < x_{min}$, set $X = \emptyset$.

All of this applies nicely to our scenario, as we can systematically apply these rules to each constraint step by step. The resulting polytope, based on the interval approach, is box-shaped. This shape further reduces our set of possible state variables and control inputs.

Cylindrical Algebraic Decomposition

The second approach is to use Cylindrical Algebraic Decomposition (CAD) to find the inner polytope. This method is computationally more expensive but provides a more accurate result. The idea is to find an equivalent formula without quantifiers for a given formula containing quantifiers. This can be done using CAD.

CAD is a method used in computer algebra for solving systems of polynomial equations and inequalities. Given a set of polynomial equations and inequalities, CAD decomposes the space into a finite number of cylindrical cells. Each cell is described by a sequence of polynomial inequalities and has a constant truth value over the input set of polynomial equations and inequalities. This way, one only needs to pick one point from each cell to check the truth value of the input set. The number of cells grows doubly exponentially with the number of variables in the input set. Several implementations of CAD exist.

We will illustrate how the algorithm works, but it is beyond the scope of this work to explain the implementations in detail. Instead, we provide a basic example to illustrate the algorithm without delving into the details. You can find more information in the literature [3, 4].

Example

To illustrate the process of eliminating $\forall$ quantifiers, consider the following problem:

$$\forall x, x^2 + bx + 1 \geq 0$$

Since quantifier elimination is typically performed on existential quantifiers, we first solve the problem for:

$$\exists x, x^2 + bx + 1 < 0$$

Once we have the solution for this problem, we can take the complement over $\mathbb{R}$ to obtain the solution for the original problem. We start by applying CAD to the polynomial $x^2 + bx + 1$, which results in 7 cells, as illustrated in Figure 3.1. Most of the cells are open, and if the edge of the cell is part of it, it is depicted with the same color but dashed.

We now need to check the truth value of the polynomial inequality $x^2 + bx + 1 < 0$ for each cell by picking a random sample and evaluating the inequality. The cells where the inequality holds true are colored in green and then projected onto the b-axis. The

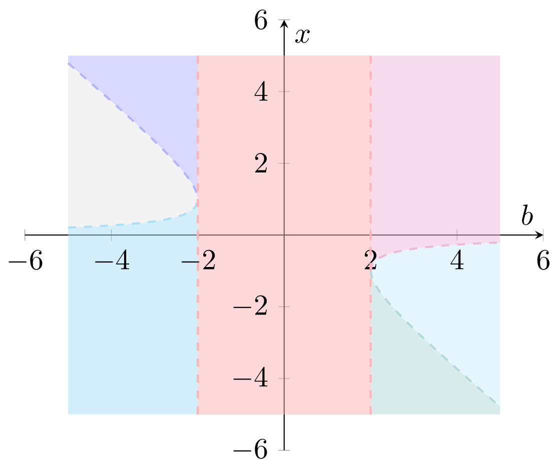


Figure 3.1: Illustrating the cells with shaded regions.

resulting intervals are the solution to the existential quantifier elimination, which is $(-\infty, -2) \cup (2, \infty)$. As an input to the algorithm, one must define an order of precedence for the variables. In its first phase, the algorithm projects the space iteratively from $\mathbb{R}^n$ to $\mathbb{R}^{n-1}$ until it reaches $\mathbb{R}^1$. This is done by removing one of the remaining variables in each iteration, starting with the last in the order of precedence. Therefore, if b is defined as the first variable, the sets of polynomial inequalities for each cell will always contain an interval for b, which corresponds to the projection on the axis. This allows us to directly read the resulting intervals, which is shown in Figure 3.2.

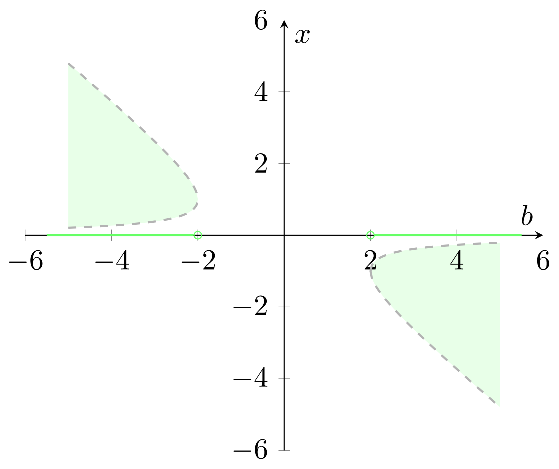


Figure 3.2: Illustrating the remaining cells.

Since $(-\infty, -2) \cup (2, \infty)$ is the solution for $\exists x, x^2 + bx + 1 < 0$, the solution for the original problem is obtained by taking the complement of this set:

$$\forall x, x^2 + bx + 1 \geq 0 \iff b \in [-2, 2]$$

Using this technique for eliminating quantifiers from formulas, we can apply it to find an inner polytope.

Comparison of Interval Fitting and CAD Approaches

In our benchmarks, the first approach, despite its simplicity, performed remarkably well. We used the following parameters for the benchmarks:

- $C(s) \in \left[-\frac{1}{200}, 0\right]$, $s \in [0, 10]$, $n \in [0, 2]$
- $v_x \in [0, 10]$, $v_y \in [-2, 2]$, $a_x \in [-3, 6]$, $a_y \in [-4, 4]$
- $\dot{\psi} \in [-5, 5]$, $a_\psi \in [-2, 2]$

The results are shown in the following figures 3.3.

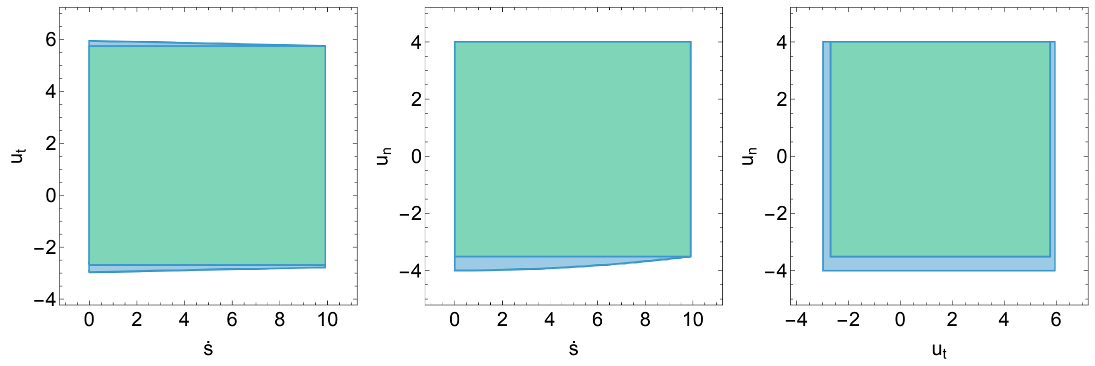


Figure 3.3: In Green our first approach and in Blue using CAD.

In conclusion, eliminating the quantifiers with the first approach leads to a near-optimal result, comparable to the one achieved using the second approach with CAD. However, using CAD has a significant downside that we have not yet mentioned. It is not guaranteed that the resulting formula is convex. Typically, you will encounter disjunctions of polynomial inequalities, which cannot be handled by a convex solver without resorting to integer programming or an equivalent approach. It is also not guaranteed that each polynomial inequality follows the DCP rules, even if the set described by the resulting formula is convex. Additional techniques would be required to use the second approach for our planner.

In cases where the resulting set is convex, we used a sampling approach to obtain an inner approximation described by half-spaces. This results in a sequence of linear constraints that all must be satisfied, thus losing a small proportion of the original set. Consequently, the difference between our first approach and the second approach becomes even smaller. Additionally, we end up with double or triple the number of constraints on each state variable and each control input, which may lead to slower solver times.

Overall, while the CAD approach provides a more accurate result, the first approach offers a good balance between computational efficiency and accuracy, making it a viable option for practical applications.

Inner Polytope

We have managed to construct a realistic model, which allows us to model any constraints arising either from the vehicle dynamics or the environment, such that they are not only confirm with the DCP rules but also linear.

Our inner polytope from the problem 3.1.2 is now given by:

$$\underline{\mathcal{C}} = \underline{\tilde{\mathcal{C}}} \times [\underline{s}, \overline{s}] \times [\underline{n}, \overline{n}] \times [\underline{\dot{n}}, \overline{\dot{n}}] \tag{3.36}$$

with $\underline{\tilde{\mathcal{C}}}$ resulting from one of the presented ∀-elimination approach.

Limitations and Outlook

While the ∀-elimination approach provides a computationally efficient method to find intervals for the variables of interest, it can be quite restrictive for several reasons:

- **Conservativeness:** The approach tends to be conservative because it ensures that the constraints hold for all possible values within the intervals. This often leads to smaller intervals, which may exclude feasible solutions that could be considered by less conservative methods.
- **Dependence on Affine Functions:** The first method relies on the assumption that the function $f(x, y)$ is affine in x and all variables in y are bounded. If this assumption does not hold, the approach may not be applicable.

Overall, while the ∀-elimination approach is useful for its simplicity and computational efficiency, it may lead to overly restrictive solutions that do not fully exploit the feasible region.

Consider a scenario with a tight turn followed by a long straight road. In such cases, the model will restrict $\dot{s}$ to an interval that is valid for both the tight turn and the straight road. Consequently, the model will find a solution, but it will not be able to drive fast on the straight road, even though it is possible to drive faster on the straight road than on the tight turn.

To address this issue, we can introduce segments of the road, one for the straight road and one for the tight turn. We can independently construct the coupling constraints set for each segment. However, this introduces a new problem: how to switch between the segments. Our solution involves using the current vehicle velocity to predict when the vehicle will reach the end of the segment. Knowing the vehicle's current position, velocity, and the distance to the end of the segment, we can calculate the time it will take to reach the end.

Further, both approaches do not consider the possibility of achieving a larger feasible set by restricting $\dot{n}$ to a smaller interval. The first approach handles the problem by using the bounds on s, n, and $\dot{n}$ to find the intervals for $\dot{s}$, which are then used for u_t and finally for u_n. However, changing the order may lead to better results for desired driving behavior.

Given the simplicity of the first approach, we implemented a non-linear program that defines the relationships between the intervals with variables as upper and lower bounds. By adding constraints, we can define an objective that models certain driving behaviors. For example, one might want to be able to slow down as quickly as possible or maximize the upper speed limit. The latter objective leads to the following intervals:

$$\begin{aligned} 0 &\leq s \leq 10 \\ 0 &\leq n \leq 2 \\ 0 &\leq \dot{s} \leq 10 \\ -2 &\leq \dot{n} \leq 2 \\ -2.9 &\leq u_t \leq 5.9 \\ -4 &\leq u_n \leq 3.75 \end{aligned}$$

(a) Initial Approach

$$\begin{aligned} 0 &\leq s \leq 10, \\ 0 &\leq n \leq 2, \\ 0 &\leq \dot{s} \leq 10.05 \\ -2 &\leq \dot{n} \leq 2 \\ -2.899 &\leq u_t \leq 5.929 \\ -4 &\leq u_n \leq 3.746 \end{aligned}$$

(b) Using Non-Linear Programming

Figure 3.4: Comparison of two sets of intervals for state variables and control inputs.

As you can see, the intervals on the right are preferable, as they only slightly reduce longitudinal acceleration while allowing for higher speeds.

In conclusion, we have successfully developed our first vehicle model for motion planning that can be solved using a convex solver. Next, we will introduce a transformation that maps the state variables and control inputs of the planning model to a steering angle, which can be used to control a vehicle based on the equations from [1]. A similar approach is demonstrated in [5], which served as an additional inspiration.

3.1.4 Determining the Steering Angle

Typically, a vehicle is controlled through throttle, brakes, and a steering angle. To incorporate these controls, we need to move away from visualizing our model as a box aligned with the road. Instead, we will treat the model as a point and define its orientation based on its velocity. Using the equations (3.3) and (3.4) with $v_y = 0$, we can solve for v_x.

$$v := v_x = \sqrt{(1 - nC(s))^2 \dot{s}^2 + \dot{n}^2} \tag{3.37}$$

Dividing $\dot{n}$ by $\dot{s}$ yields:

$$\frac{\dot{n}}{\dot{s}} = (1 - nC(s)) \tan(\xi) = (1 - nC(s)) \tan(\psi - \theta) \tag{3.38}$$

which we can solve for ψ to get the orientation of the vehicle.

$$\psi = \theta + \arctan(\frac{\dot{n}}{\dot{s}(1 - nC(s))}) \tag{3.39}$$

Using the state variables and g from (3.27), we can calculate $a_{x,tn}$ and $a_{y,tn}$ from (3.7) and (3.8), respectively. By substituting these values into equations (3.5) and (3.6), and setting $a_y = 0$, we can determine the longitudinal acceleration and the change in orientation. Additionally, we assume $|\xi| \leq \frac{\pi}{2}$ to ensure that $\cos \xi \neq 0$.

$$\dot{\psi} = \frac{a_{y,tn} - \tan(\xi) a_{x,tn}}{v(\tan(\xi) \sin(\xi) + \cos(\xi))} \tag{3.40}$$

$$a := a_x = \frac{a_{x,tn} + v\dot{\psi} \sin(\xi)}{\cos \xi} \tag{3.41}$$

Our Bicycle models (2.17) enables us to calculate the steering angle.

$$\delta = \arctan(l_{wb} \frac{\dot{\psi}}{v}) \tag{3.42}$$

With those equations, we can define a transformation.

$$T(x_1, \tilde{u}) = [p_x, p_y, \psi, \dot{\psi}, v, a, \delta] \tag{3.43}$$

3.1.5 Final Model

Our final model is represented by the following tuple.

$$M_{pm} = (x_1, \tilde{u}, f_1, \underline{\mathcal{C}}, T) \tag{3.44}$$

3.2 Motion Planning using Bicycle Model

Instead of using the point mass model, we can employ the bicycle model to represent vehicle dynamics more accurately. We have already introduced the bicycle model with a steering angle and an orientation. Now, we will combine this model with the concepts from the previous chapter. Our objective is to represent the state variables in the road-aligned frame. The state and control variables of the bicycle model, defined in the global coordinate system, are given in equations (2.11) and (2.12).

3.2.1 Transforming Global Cartesian Coordinates to Frenet Frame

In this section, we derive the state transition model in the Frenet frame.

We begin by considering the dynamics of the bicycle model in the global coordinate system, which are described by (**??**)-(2.17).

To express vehicle motion in the Frenet frame, we define the deviation from the reference path using the lateral displacement n and the orientation error $\xi = \psi - \theta$, where θ is the heading of the reference path. The path curvature $C(s)$ relates to its arc length parameter s as $\dot{\theta} = C(s)\dot{s}$.

Using the previously derived equations (3.3) and (3.4), we obtain the state transition equations in the Frenet frame:

$$\dot{s} = \frac{v \cos\xi}{1 - nC(s)} \tag{3.45}$$

$$\dot{n} = v \sin\xi \tag{3.46}$$

$$\dot{\xi} = \dot{\psi} - C(s)\dot{s} \tag{3.47}$$

By integrating these equations with the bicycle model, we derive a complete state transition model in the Frenet frame. The state variables and control inputs for the bicycle model in the Frenet frame are defined in (3.48) and (3.49).

$$x_{kst} = \begin{bmatrix} s & n & \xi & v & \delta \end{bmatrix}^T \tag{3.48}$$

$$u_{kst} = \begin{bmatrix} a & v_\delta \end{bmatrix}^T \tag{3.49}$$

The dynamics of model are described by (3.50).

$$f_{kst}(x_{kst}, u_{kst}) = \begin{bmatrix} \dfrac{v\cos\xi}{1 - nC(s)} \\ v\sin\xi \\ \dfrac{1}{l_{wb}} v\tan\delta - C(s)\dot{s} \\ a \\ v_\delta \end{bmatrix}. \tag{3.50}$$

In the following section, we will present an approach how the model dynamics can be approximated to apply to the DCP rules.

3.2.2 Model Dynamics Approximation

For this model, we will stick to the body-fixed control inputs, which keeps the coupling constraints convex. Instead, we aim to linearize the model dynamics using two new techniques. These techniques allow us to maintain the constraints without shifting, as was necessary in the previous chapter. This ensures a more accurate and computationally efficient representation of the vehicle's motion.

To simplify the model, we make the following assumption: $nC(s)$ is close to zero. This is valid since n represents the vehicle's lateral position relative to the reference path, and $C(s)$ is the curvature of the reference path, which is typically small enough for this assumption to hold. We will analyze the terms that introduce non-linearity into the model.

Non-Linear Terms

The state transition model contains four non-linear terms. We will linearize these terms using appropriate approximations.

$$\frac{v\cos\xi}{1 - nC(s)}, v\sin\xi, v\tan\delta, C(s)\dot{s}$$

Given our assumption that $nC(s) \approx 0$, we can simplify the first term as follows:

$$\frac{v\cos\xi}{1 - nC(s)} \approx v\cos\xi$$

First Order Taylor Polynomial

To linearize the trigonometric terms, we use the first-order Taylor polynomial approximation around a reference point. The first-order Taylor expansion of a function $f(x)$

around a point x_0 is given by:

$$f(x) \approx f(x_0) + \frac{df}{dx}(x_0)(x - x_0)$$

Using the first-order Taylor polynomial to approximate the trigonometric functions sin, cos, and tan around the reference points ξ_0 and δ_0, we obtain the following linearizations:

$$\begin{aligned}
\cos(\xi) &\approx \cos(\xi_0) - \sin(\xi_0)(\xi - \xi_0) \\
\sin(\xi) &\approx \sin(\xi_0) + \cos(\xi_0)(\xi - \xi_0) \\
\tan(\delta) &\approx \tan(\delta_0) + \frac{1}{\cos^2(\delta_0)}(\delta - \delta_0)
\end{aligned}$$

These approximations are known as small angle approximations, which are valid when the angles ξ and δ are close to their reference values. In vehicle dynamics, it is often reasonable to assume that the heading alignment error ξ and the steering angle δ do not change rapidly, especially when the vehicle is closely following a reference path. This allows us to simplify the trigonometric functions using their first-order Taylor expansions.

By substituting these approximations into the state transition model, we obtain the following terms:

$$\begin{aligned}
&v(\cos(\xi_0) - \sin(\xi_0)(\xi - \xi_0)) \\
&v(\sin(\xi_0) + \cos(\xi_0)(\xi - \xi_0)) \\
&v(\tan(\delta_0) + \frac{1}{\cos^2(\delta_0)}(\delta - \delta_0))
\end{aligned}$$

Since our reference values ξ_0 and δ_0 are treated as constants during planning, the only remaining non-linear terms are:

$$v\xi, v\delta, C(s)\dot{s}$$

These are known as bilinear terms, which occur when the product of two variables appears in the equations, rendering the system non-linear. Since $C(s)$ is a function of s, we will introduce an additional assumption. We have previously demonstrated that our road can be modeled in segments, a technique we employed for the Point Mass Model to achieve less conservative coupling constraints. In this context, we will apply the same approach to linearize our dynamics.

Assumption: Piece Wise Linear Curvature

We assume that the curvature can be approximated as a piece-wise linear function.

$$C(s) = \begin{cases} a_1 s + b_1 & \text{if } s \in [s_0, s_1] \\ a_2 s + b_2 & \text{if } s \in [s_1, s_2] \\ \vdots & \\ a_n s + b_n & \text{if } s \in [s_{n-1}, s_n] \end{cases}$$

This transformation reduces the nonlinear term $C(s)\dot{s}$ to a bilinear term $s\dot{s}$, which still requires linearization.

In the next section, we will introduce a relaxation method to achieve this.

3.2.3 Convex Relaxation of Bilinear Terms

To handle bilinear terms of the form xy, we introduce a new variable w and apply the McCormick relaxation. The McCormick relaxation is a technique that linearizes bilinear terms by introducing auxiliary variables and constraints. This technique allows us to represent the bilinear terms as a set of linear constraints, which can be solved efficiently using convex optimization methods. This relaxation only works if the variables x and y are bounded:

$$\underline{x} \leq x \leq \overline{x}, \qquad \underline{y} \leq y \leq \overline{y}$$

with constants $\underline{x}, \overline{x}, \underline{y}, \overline{y} \in \mathbb{R}$.

We introduce an auxiliary variable w to approximate the bilinear term xy. Linear constraints are applied to bound w and ensure it accurately represents the bilinear term xy. These constraints are derived from the bounds of the variables x, y, and the bilinear term xy, as shown below:

$$\begin{aligned} w &\geq \underline{x}y + x\underline{y} - \underline{x}\underline{y}, \\ w &\geq \overline{x}y + x\overline{y} - \overline{x}\overline{y}, \\ w &\leq \overline{x}y + x\underline{y} - \overline{x}\underline{y}, \\ w &\leq \underline{x}y + x\overline{y} - \underline{x}\overline{y}. \end{aligned}$$

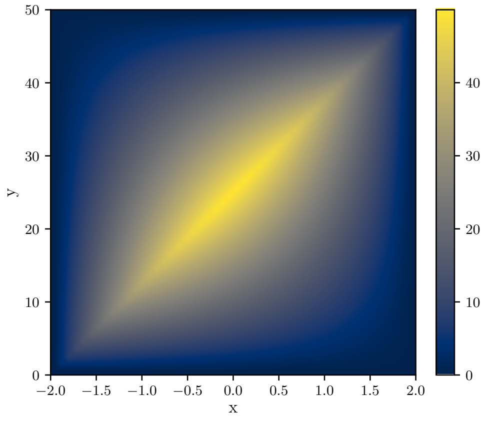
The idea behind these constraints is to create a convex lower bound and a concave upper bound for the bilinear term xy. For example, the first bound is constructed as follows:

$$a = (x - \underline{x}) \geq 0, b = (y - \underline{y}) \geq 0$$

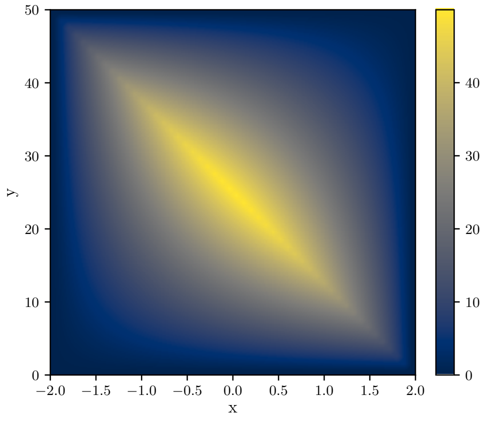
Since a and b are both non-negative, we can multiply them to get and obtain our first lower bound:

$$ab = xy - \underline{x}y - x\underline{y} + \underline{x}\underline{y} \geq 0 \iff xy \geq \underline{x}y + x\underline{y} - \underline{x}\underline{y}$$

To derive all possible lower and upper bounds, we can apply this pattern to $a \in \{x - \underline{x}, \overline{x} - x\}$ and $b \in \{y - \underline{y}, \overline{y} - y\}$. For any of the four possible combinations of a and b, we get $ab \geq 0$ and can therefore establish either an upper or lower bound for xy.



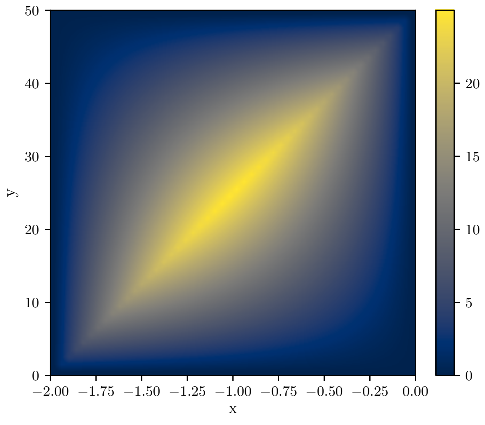
(a) Difference to the upper bound



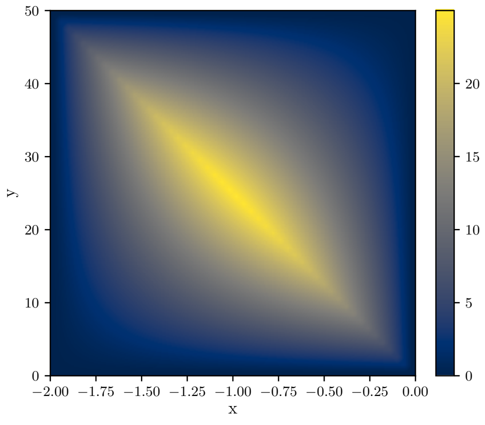
(b) Difference to the lower bound

Figure 3.5: McCormick relaxation bounds for the bilinear term xy.

Figure 3.5a shows the difference to the upper bound, while Figure 3.5b shows the difference to the lower bound for the range $-2 \leq x \leq 2$ and $0 \leq y \leq 50$. It is evident that the bounds become tighter as x and y approach their respective limits.



(a) Difference to the upper bound



(b) Difference to the lower bound

Figure 3.6: McCormick relaxation with tighter bounds on x.

Figures 3.6a and 3.6b show the results when x is more tightly bounded, specifically

$-2 \leq x \leq 0$. It is also clear that the maximum deviation is significantly reduced compared to the previous scenario, demonstrating that tighter bounds lead to a more accurate relaxation.

Improve Bounds

To refine the McCormick relaxation bounds, we enforce tighter limits on the variables in the bilinear terms. Narrower ranges yield a more accurate approximation of these products by reducing conservatism in the relaxation.

For example, if the variable x is always in the range $-1 \leq x \leq 1$ and y is within $0 \leq y \leq 25$, applying these stricter bounds enhances the approximation accuracy.

For the vehicle velocity, we can derive tighter bounds using the current velocity and the known acceleration limits. Let v_0 be the current velocity, $\underline{a}$ the minimum acceleration, and $\overline{a}$ the maximum acceleration. Then the velocity at the next time step v_1 is bounded by:

$$v_1 \in \left[v_0 + \underline{a}\Delta t,\ v_0 + \overline{a}\Delta t\right]$$

where Δt is the time step.

By iteratively applying these bounds for subsequent time steps, we can obtain increasingly precise constraints for the velocity. This leads to a tighter and more efficient McCormick relaxation for the bilinear terms.

$$v_2 \in [v_0 + 2\underline{a}\Delta t, v_0 + 2\overline{a}\Delta t]$$

In summary, enhancing the McCormick relaxation bounds involves the following steps: 1. Analyzing the problem in detail to determine accurate and realistic variable limits. 2. Employing tighter bounds for the variables that appear in the bilinear terms. 3. Using these refined limits to achieve a more precise approximation of the bilinear expressions.

This strategy ensures that the McCormick relaxation delivers a more accurate and efficient representation of the bilinear terms, which is critical for the precise modeling and control of vehicle dynamics.

3.2.4 Final Model

The final approximated dynamics model in the Frenet frame, which integrates the linearized non-linear terms with the McCormick relaxation for the bilinear terms, is given by:

$$\tilde{f}_{kst}(x_{kst}, u_{kst}) = \begin{bmatrix} v(\cos(\xi_0) + \sin(\xi_0)\xi_0) - \sin(\xi_0) w_{v,\xi} \\ v(\sin(\xi_0) - \cos(\xi_0)\xi_0) + \cos(\xi_0) w_{v,\xi} \\ \frac{v}{l_{wb}}(\tan(\delta_0) - \frac{\delta_0}{\cos^2(\delta_0)}) + \frac{w_{v,\delta}}{\cos^2(\delta_0)} - a_{i(s)} w_{s,\dot{s}} - b_{i(s)}\dot{s} \\ a \\ v_\delta \end{bmatrix} \tag{3.51}$$

Here, the auxiliary variables $w_{v,\xi}$, $w_{v,\delta}$, and $w_{s,\dot{s}}$ are introduced via the McCormick relaxation to linearize the bilinear terms $v\xi$, $v\delta$, and $s\dot{s}$, respectively. The curvature is modeled as a piece wise linear function, where $a_{i(s)}$ and $b_{i(s)}$ denote the slope and intercept for the interval $[s_{i-1}, s_i]$, selected according to the current value of s. For planning purposes, the index $i(s)$ will be treated as a constant at each time step by predicting the relevant road segment. The following additional constraints enforce the McCormick relaxation:

$$\begin{aligned}
w_{v,\xi} &\geq \underline{v}\xi + v\underline{\xi} - \underline{v}\underline{\xi}, & w_{v,\delta} &\geq \underline{v}\delta + v\underline{\delta} - \underline{v}\underline{\delta}, & w_{s,\dot{s}} &\geq \underline{s}\dot{s} + s\underline{\dot{s}} - \underline{s}\underline{\dot{s}}, \\
w_{v,\xi} &\geq \overline{v}\xi + v\overline{\xi} - \overline{v}\overline{\xi}, & w_{v,\delta} &\geq \overline{v}\delta + v\overline{\delta} - \overline{v}\overline{\delta}, & w_{s,\dot{s}} &\geq \overline{s}\dot{s} + s\overline{\dot{s}} - \overline{s}\overline{\dot{s}}, \\
w_{v,\xi} &\leq \overline{v}\xi + v\underline{\xi} - \overline{v}\underline{\xi}, & w_{v,\delta} &\leq \overline{v}\delta + v\underline{\delta} - \overline{v}\underline{\delta}, & w_{s,\dot{s}} &\leq \overline{s}\dot{s} + s\underline{\dot{s}} - \overline{s}\underline{\dot{s}}, \\
w_{v,\xi} &\leq \underline{v}\xi + v\overline{\xi} - \underline{v}\overline{\xi}. & w_{v,\delta} &\leq \underline{v}\delta + v\overline{\delta} - \underline{v}\overline{\delta}. & w_{s,\dot{s}} &\leq \underline{s}\dot{s} + s\overline{\dot{s}} - \underline{s}\overline{\dot{s}}.
\end{aligned}$$

Figure 3.7: McCormick relaxation constraints for the bilinear terms.

Coupling Constraints

The coupling constraints limit the state and control variables to ranges that reflect vehicle performance and road conditions. In our optimization setup, these constraints help maintain realistic and consistent bounds. They are specified as follows:

$$s \in [\underline{s}, \overline{s}], \quad n \in [\underline{n}(s), \overline{n}(s)], \quad \xi \in [\underline{\xi}, \overline{\xi}], \quad v \in [\underline{v}, \overline{v}], \quad \delta \in [\underline{\delta}, \overline{\delta}] \tag{3.52}$$

$$v_\delta \in [\underline{v_\delta}, \overline{v_\delta}], \quad a \in \left[\underline{a}_{x,b},\ \overline{a}_{x,b} \min\left\{1, \frac{v_S}{v}\right\}\right] \tag{3.53}$$

where v_S is a parameter representing a scaling factor, which is used to encounter limiting engine power and breaking power.

Finally, we must consider the friction circle constraint.

$$\sqrt{a^2 + (\frac{v^2}{l_{wb}} \tan(\delta))^2} \leq a_{max} \tag{3.54}$$

We define $v^* := max\{|\underline{v}|, |\overline{v}|\}$ and $\delta^* := max\{|\underline{\delta}|, |\overline{\delta}|\}$. Utilizing that $\tan(\delta) \leq \frac{\tan(\delta^*)}{\delta^*}\delta$ leads to a bit more conservative constraint.

$$a^2 + (\frac{1}{l_{wb}} \frac{\tan(\delta^*)}{\delta^*})^2 v^4 \delta^2 \leq a_{max}^2 \tag{3.55}$$

Next, we derive an upper bound w for the term $v^4\delta^2$, which will be used to further simplify our friction circle constraints.

Upper Bound for $v^4\delta^2$

We are applying a diamond shaped set, by introducing the hyperplanes:

$$d_v v + d_\delta \delta \leq w \tag{3.56}$$

$$d_v v - d_\delta \delta \leq w \tag{3.57}$$

$$-d_v v + d_\delta \delta \leq w \tag{3.58}$$

$$-d_v v - d_\delta \delta \leq w \tag{3.59}$$

where $d_v = \frac{1}{v^*}$ and $d_\delta = \frac{1}{\delta^*}$.

To determine an appropriate value for w, we set (3.55) as an equality and solve for δ.

$$\delta = h(a)\frac{1}{v^2} \tag{3.60}$$

where

$$h(a) := \frac{l_{wb}\delta^*}{\tan(\delta^*)}\sqrt{a_{max}{}^2 - a^2}$$

We want to push the hyperplane as close the bounds as possible, such that is tangent to the bounds, which lets us solve for w by the deriving (3.60) after v and setting it equal to slope of the first hyperplane.

$$\frac{\partial \delta}{\partial v} = -2h(a)\frac{1}{v^3} \overset{!}{=} -\frac{d_v}{d_\delta} \iff v = \sqrt[3]{2h(a)\frac{d_\delta}{d_v}} \tag{3.61}$$

Substituting the value of v into (3.60), we get:

$$\delta = h(a)\left(2h(a)\frac{d_\delta}{d_v}\right)^{-\frac{2}{3}} \tag{3.62}$$

These values for v and δ must lie on the first hyperplane, resulting in:

$$w = w(a) = d_v \sqrt[3]{2h(a)\frac{d_\delta}{d_v}} + d_\delta h(a) \left(2h(a)\frac{d_\delta}{d_v}\right)^{-\frac{2}{3}} \tag{3.63}$$

To obtain a convex upper bound for w, we fit an irrational function of the form:

$$w(a) \approx \tilde{w}(a) := c_1 - \frac{1}{(|a| - c_2)^{2n}} \tag{3.64}$$

where $c_1 = w(0)$ and $c_2 = a_{max} + c_1^{-\frac{1}{2n}}$.

Based on our vehicle parameters $\delta^* = 0.910$, $l_{wb} = 2.4$, and $a_{max} = 11.5$, we evaluated this approach on a velocity limit of $v^* = 14$. Figure 3.8 illustrates the function $w(a)$ and its approximation with $n = 2$ as a function of the acceleration a.

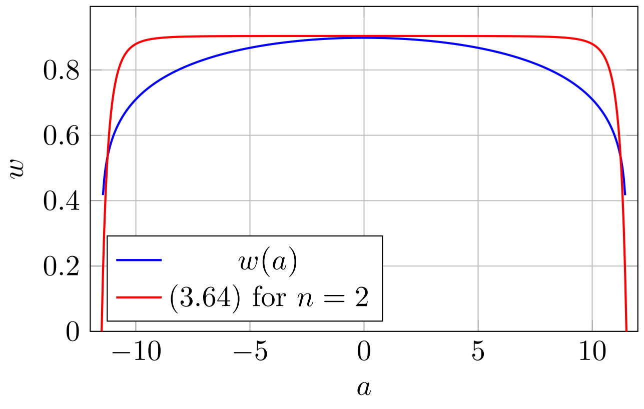


Figure 3.8: Plot of w versus a and its approximation.

The following figures 3.9 visualize the friction circle (3.54), its tighter constraint (3.55) and the resulting diamond-shaped bounds for v and δ for a fixed value of a.

In conclusion, this approach constructs a convex approximation of the friction circle and allows for weighting either v or δ by setting v^*. The final constraint can be stated as follows:

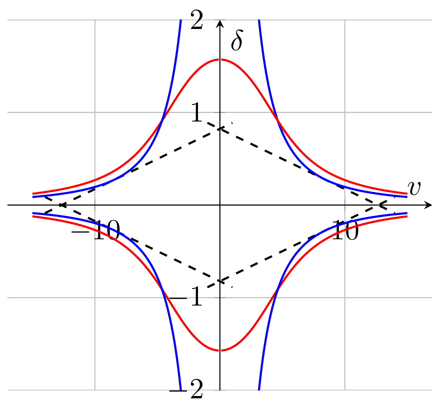
$$a^2 + (\frac{1}{l_{wb}}\frac{\tan(\delta^*)}{\delta^*})^2 w \leq a_{max}^2 \tag{3.65}$$

where w is new introduced auxiliary variable, which is bounded by:

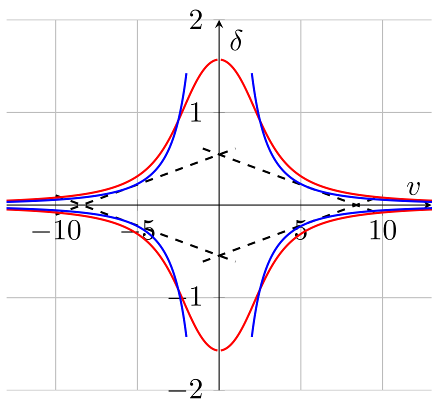
$$0 \leq w \leq \tilde{w}(a) \tag{3.66}$$

Our final model is now complete and ready for use in the optimization process. It is represented by the tuple

$$M_{kst} = (x_{kst}, u_{kst}, \tilde{f}_{kst}, \{w_{v,\xi}, w_{v,\delta}, w_{s,\dot{s}}, w\}, C) \tag{3.67}$$



(a) Plot of the resulting diamond at $a = 0$.



(b) Plot of the resulting diamond at $a = 11$.

- - - Diamond Constraint
—— Tighter Friction Constraint
—— Friction Constraint

Figure 3.9: Comparison of the resulting diamond constraints for different acceleration values.

where C consists of the coupling constraints (3.52), (3.53), the McCormick bounds (3.7), the introduced hyperplanes (3.56)-(3.59), and our approximated friction circle (3.65) with (3.66).

3.3 Modeling Techniques

In the previous chapters, we have explored various modeling techniques that are essential for solving complex optimization problems. We began by discussing the decoupling of variables using quantifier elimination, a powerful method that simplifies the problem by reducing the number of variables involved. This technique is particularly useful in scenarios where the relationships between variables are intricate and non-linear.

Next, we examined convex relaxations, specifically focusing on McCormick envelopes for bilinear terms. Convex relaxations are important for transforming non-convex problems into convex ones, which are easier to solve. McCormick envelopes provide a way to approximate bilinear terms with convex functions, thereby enabling the use of efficient convex optimization algorithms.

In this section, we present a collection of commonly used modeling methods for convex programming. These methods serve as a practical guide for formulating and solving convex optimization problems.

3.3.1 Soft Constraints

Soft constraints are used in optimization problems where certain constraints can be violated to some extent, but with a penalty. This is in contrast to hard constraints, which must be strictly satisfied. Soft constraints are particularly useful in real-world scenarios where it is often impractical to meet all constraints perfectly.

To incorporate soft constraints into a convex optimization problem, we introduce slack variables and a penalty term in the objective function. The slack variables measure the extent of constraint violation, and the penalty term ensures that violations are minimized.

Consider a constraint of the form $g(x) \leq 0$. To make this constraint soft, we introduce a slack variable $s \geq 0$ and modify the constraint to $g(x) \leq s$. We then add a penalty term λs to the objective function, where λ is a positive weight that controls the trade-off between minimizing the original objective and satisfying the constraint.

The modified optimization problem can be written as:

$$\begin{aligned} \min_{x,s} \quad & f(x) + \lambda s \\ \text{subject to} \quad & g(x) \leq s \\ & s \geq 0 \end{aligned}$$

By adjusting the value of λ, we can control the degree to which the soft constraint is enforced. A larger λ places more emphasis on satisfying the constraint, while a smaller λ allows for greater flexibility in violating the constraint.

3.3.2 Auxiliary Variables

Auxiliary variables can be used for modeling in many ways. In our models we are the defining the road with as a function over s the distance along the road. One common part objective may be to minimize the offset to the center of the road. The first formulation that may come to mind is:

$$\min g(x,u) + \left(n - \frac{\overline{n}(s) - \underline{n}(s)}{2}\right)^2$$

This is a valid formulation, but it is not convex. Instead, we are using different approach to formulate the offset to the center of the road.

$$\min \{\overline{n}(s) - n, n - \underline{n}(s)\}$$

which gives us the distance to the closer boundary of the road. This formulation is concave, if $\overline{n}(s)$ is concave and $\underline{n}(s)$ is convex. By introducing the auxiliary variable d which is constrained by:

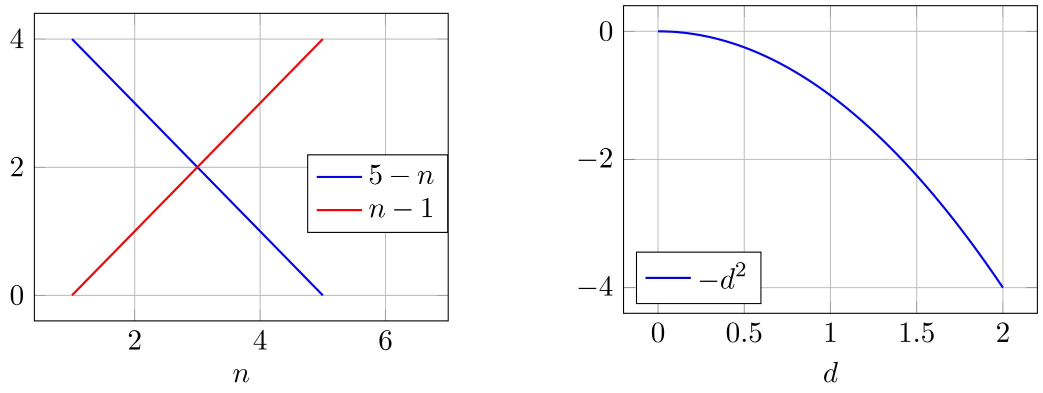
$$0 \leq d \leq \min \{\overline{n}(s) - n, n - \underline{n}(s)\}$$

we can reformulate the objective as:

$$\min g(x,u) - d^2$$

This formulation is convex and can be solved efficiently.

To visualize these formulations, we can plot them using a constant value for $\overline{n}(s)$ and $\underline{n}(s)$ in Figure 3.10. Let's assume $\overline{n}(s) = 5$ and $\underline{n}(s) = 1$.



(a) Distance to Road Boundaries (b) Objective Function

Figure 3.10: Plots the distance to the road boundaries and the objective function.

While $-\min(5-x, x-1)$ is a convex function, it is piece wise linear, which can lead to difficulties in optimization. The benefit of using an auxiliary variable in this context is that it allows us to transform a piece wise linear and potentially non-differentiable objective function into a smooth and differentiable convex function. This transformation simplifies the optimization process, making it more efficient and reliable. Specifically, by introducing the auxiliary variable d and reformulating the objective as $\min g(x,u) - d^2$, we obtain a function that is easier to handle with gradient-based optimization algorithms, which rely on smoothness and differentiability to find optimal solutions effectively.

3.3.3 Penalty Methods

Penalty methods are another approach to handle constraints in optimization problems. These methods incorporate constraints into the objective function by adding a penalty term that increases the objective value when constraints are violated. This approach transforms a constrained optimization problem into an unconstrained one, which can be easier to solve.

Consider an optimization problem with a constraint $g(x) \leq 0$. In a penalty method, we modify the objective function to include a penalty term $P(g(x))$ that penalizes constraint violations. A common choice for the penalty term is a quadratic function, such as $P(g(x)) = \mu \max(0, g(x))^2$, where μ is a positive penalty parameter.

The modified optimization problem can be written as:

$$\min_{x} \quad f(x) + \mu \max(0, g(x))^2$$

By adjusting the value of μ, we can control the severity of the penalty for constraint violations. A larger μ places more emphasis on satisfying the constraint, while a smaller μ allows for greater flexibility in violating the constraint.

Penalty methods are particularly useful when dealing with complex constraints that are difficult to handle directly. By incorporating these constraints into the objective function, we can leverage efficient unconstrained optimization algorithms to find solutions.

3.3.4 Conclusion

In this chapter, we have explored various modeling techniques that are essential for solving complex optimization problems. We discussed the use of soft constraints, auxiliary variables and penalty methods, each of which offers unique advantages for handling different types of constraints and objectives. By understanding and applying these techniques, we can formulate and solve optimization problems more effectively, leading to better solutions and improved performance in practical applications.

4 Evaluation

In this chapter, we evaluate the performance of our proposed method using a set of benchmark problems. We introduced two models, M_{pm} (3.44) and M_{kst} (3.67), each defined by its own dynamic equations and constraints on state variables and control inputs.

To enable realistic driving tests, we must address key implementation details, which we will discuss next. Additionally, a simulation model is required to assess the performance of the planned trajectory. This simulation allows us to recreate real-world driving scenarios and evaluate our models under practical conditions

4.1 Implementation Details

4.1.1 Road Segments

For our point mass model, we want to split up the road into segments, based on mainly their curvature. As already discussed this approach allows the model to have a larger search space during planning. The introduced kinematic single track model assumes the curvature to be linear, which is only practical for the road topology if we allow a piece wise linear curvature and select the current piece. We can easily add other road topology constraints to be dependent by the current segment, such as the road width.

Our planner operates on a time horizon, divided into discrete time steps $\{t_i\}_{i=1,\ldots,n}$. For each time point, we seek the state and the transition to the next time point, controlled by the input. The transition is constrained by the dynamics of the model, the state variables, and the control input through coupling constraints. To make the dynamics and coupling constraints dependent on the road segment, we need to provide the current road segment for each time point.

Given the start and end of each road segment $\{[s_{i-1}, s_i]\}_{i=1,\ldots,m}$ and the current position s of the vehicle, with a reference velocity $v(t)$ which can be time-dependent, we can determine the current road segment for each time point by:

$$i_{s,\{s_i\}_{i=0,\ldots,m},v} : \{t_i\}_{i=1,\ldots,n} \to \{1,\ldots,m\}, t \mapsto i(t) = \min\left\{ j \mid s + \int_0^t v(\sigma) d\sigma \leq s_j \right\} \tag{4.1}$$

This function can be used to determine the road segment-dependent variables. We want to make not only the curvature road segment-dependent but also the road width. The road width consists of the left $\overline{n}(s)$ and right $\underline{n}(s)$ lane width. The left lane width can be concave, and the right lane width can be convex. Thus, our implementation of a road segment includes a linear curvature, a concave and convex lane width, and the length of the segment. This can be extended to include, for example, the upper velocity limits for each segment.

4.1.2 Planner

The convex discrete-time optimization problem will be solved using an external solver, which we refer to as the planner, as it provides the solution to the motion planning problem. Our trajectory planner is parameterized by a tuple $(\dim(x), \dim(u), f, \mathcal{C})$, which includes the dimensions of the state variables, the dimensions of the control inputs, the dynamics equation represented by f, and the coupling constraints represented by $\mathcal{C}$. The time horizon and discretization are given by $\{t_i\}_{i=0,\ldots,n}$.

To construct the variables and constraints for our planner, we need to define the state variables, control inputs, and the constraints that describe the transitions between states.

First, we define the state variables and control inputs for each time step t_i:

$$x(t_i) \in \mathbb{R}^{dim(x)}, u(t_i) \in \mathbb{R}^{dim(u)} \tag{4.2}$$

For each $i \in \{1, \ldots, n\}$, we define the following equality constraint:

$$x(t_i) = x(t_{i-1}) + f(x(t_{i-1}), u(t_{i-1}))(t_i - t_{i-1}) \tag{4.3}$$

These constraints, combined with the coupling constraints $\mathcal{C}$, additional constraints, and auxiliary variables for the bicycle model 3.7, define the constraints and variables for our planner.

Given the initial state $x_{initial}$, we model our initial condition with the constraint $x(t_0) = x_{initial}$. For our evaluation, we did not impose any additional constraints on the final state. Instead, we modeled our driving behavior through the objective function.

We implemented our optimization problem using Python with the 'cvxpy' library and solved it using the 'MOSEK' solver.

Next, we introduce the objectives used for our trajectory planner.

4.2 Performance Evaluation

4.2.1 Objectives

Our trajectory planner aims to minimize a cost function that represents the driving behavior we desire. The cost function is composed of several objectives, each weighted by a corresponding factor.

The primary objectives considered are control effort, deviation from the reference trajectory, and terminal state accuracy. The control effort objective aims to minimize the numerical derivatives of control inputs to ensure smooth driving, represented by the cost function:

$$J_{control} = \sum_{i=0}^{n-1} \|d_1(t_i)\|^2 \tag{4.4}$$

where $d_1(t_i) \in \mathbb{R}^{dim(u)}$ is an auxiliary variable constrained by:

$$d_1(t_i) = \frac{u(t_i) - u(t_{i-1})}{t_i - t_{i-1}}$$

The tracking objective aims to minimize the deviation from the center of the road, represented by the cost function:

$$J_{tracking} = \sum_{i=0}^{n} d_2(t_i)^2 \tag{4.5}$$

where $d_2(t_i) \in \mathbb{R}$ is an auxiliary variable constrained by:

$$0 \leq d_2(t_i) \leq \min\left\{\overline{n}(s(t_i)) - n(t_i), n(t_i) - \underline{n}(s(t_i))\right\}$$

The terminal state objective aims to minimize the deviation from a desired terminal state x_{final} at the final time step t_n, represented by the cost function:

$$J_{terminal} = \|x(t_n) - x_{final}\|^2 \tag{4.6}$$

The total cost function J is a weighted sum of these objectives:

$$J = \alpha J_{control} + \beta J_{tracking} + \gamma J_{terminal} \tag{4.7}$$

where α, β, and γ are the weights that determine the relative importance of each objective. By minimizing this cost function, our trajectory planner generates a trajectory that balances control effort, tracking the reference trajectory, and accuracy in reaching the desired terminal state.

4.2.2 Scenarios

In order to evaluate the performance of our trajectory planner, we implemented several driving scenarios. These scenarios are designed to test different aspects of the planner's capabilities. The Straight Road scenario evaluates the planner's ability to maintain a straight path with minimal control effort, ensuring smooth and efficient driving. In the

Left Turn scenario, the planner's performance is assessed based on its ability to execute a smooth left turn while adhering to the reference trajectory. The Lane Change scenario tests the planner's capability to perform a lane change maneuver safely and efficiently, highlighting its responsiveness and precision. The Slalom scenario challenges the planner to navigate through a series of closely spaced obstacles, requiring precise control and smooth transitions between maneuvers. The Elchtest, also known as the moose test, evaluates the planner's ability to perform a sudden evasive maneuver to avoid an obstacle, testing its quick decision-making and control under pressure. The Elchtest scenario can be visualized as follows:

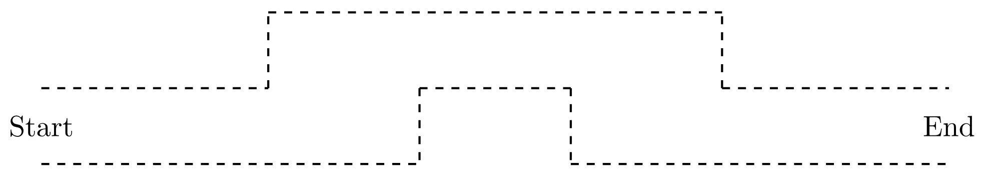


Figure 4.1: Elchtest scenario visualization

Finally, the Sharp U Turn scenario tests the planner's ability to execute a sharp U-turn, challenging its control effort and adherence to the desired terminal state.

By evaluating the planner in these diverse scenarios, we can gain a comprehensive understanding of its strengths and areas for improvement.

Table 4.1: Overview of Road Segments and Their Properties

Road Name	**Segment**	**Length**	**Curvature**	**Lane Width**	
				Start	**End**
Elchtest	1	12.0	0.000	[-1.0,1.0]	[-1.0,1.0]
	2	13.5	0.000	[-1.0,1.0]	[2.0,4.7]
	3	11.0	0.000	[2.0,4.7]	[2.0,4.7]
	4	12.5	0.000	[2.0,4.7]	[-1.0,1.0]
	5	12.0	0.000	[-1.0,1.0]	[-1.0,1.0]
Left Turn	1	235.6	0.007	[-2.0,2.0]	[-2.0,2.0]
Straight	1	180.0	0.000	[-2.0,2.0]	[-2.0,2.0]
Lane Change	1	30.0	0.000	[-2.0,2.0]	[-2.0,2.0]
	2	20.9	0.025	[-2.0,2.0]	[-2.0,2.0]
	3	20.9	-0.025	[-2.0,2.0]	[-2.0,2.0]

Continued on next page

Continued from previous page

Road Name	**Segment**	**Length**	**Curvature**	**Lane Width**	
				Start	**End**
	4	30.0	0.000	[-2.0,2.0]	[-2.0,2.0]
	1	20.0	0.000	[-2.0,2.0]	[-2.0,2.0]
	2	94.2	-0.033	[-2.0,2.0]	[-2.0,2.0]
Slalom	3	94.2	0.033	[-2.0,2.0]	[-2.0,2.0]
	4	94.2	-0.033	[-2.0,2.0]	[-2.0,2.0]
	5	20.0	0.000	[-2.0,2.0]	[-2.0,2.0]
	1	20.0	0.000	[-2.0,2.0]	[-2.0,2.0]
Feasible Curve	2	15.7	0.200	[-2.0,2.0]	[-2.0,2.0]
	3	20.0	0.000	[-2.0,2.0]	[-2.0,2.0]
	1	20.0	0.000	[-2.0,2.0]	[-2.0,2.0]
Infeasible Curve	2	8.8	0.357	[-2.0,2.0]	[-2.0,2.0]
	3	20.0	0.000	[-2.0,2.0]	[-2.0,2.0]

4.2.3 Simulation Setup

For the vehicle simulation, we employ a more sophisticated model and discretize its dynamics using the Runge-Kutta method, which offers greater accuracy compared to the forward Euler method used for trajectory planning. To ensure reproducibility, we define the model using the following state variables and control inputs:

$$x = [p_x, p_y, \delta, v, \psi, \dot{\psi}, \beta]^T, u = [a_x, v_\delta]^T$$

where p_x, p_y represent the vehicle's position coordinates, δ is the steering angle, v is the velocity, ψ is the yaw angle, $\dot{\psi}$ is the yaw rate, β is the slip angle, a_x is the longitudinal acceleration, and v_δ is the steering rate.

The model's dynamics are governed by the following equations, valid for $|v| \geq 0.1$:

$$f(x,u) = \begin{bmatrix} v\cos(\psi+\beta) \\ v\sin(\psi+\beta) \\ v_\delta \\ a_x \\ \dot{\psi} \\ \dfrac{\mu\, m}{I_z(l_r+l_f)}\Big(l_f\, C_{S,f}\Big(g\, l_r - a_x h_{cg}\Big)\,\delta \\ +\ \Big[l_r\, C_{S,r}\Big(g\, l_f + a_x h_{cg}\Big)\ -\ l_f\, C_{S,f}\Big(g\, l_r - a_x h_{cg}\Big)\Big]\,\beta\Big) \\ -\ \Big[l_f^2\, C_{S,f}\Big(g\, l_r - a_x h_{cg}\Big)\ +\ l_r^2\, C_{S,r}\Big(g\, l_f + a_x h_{cg}\Big)\Big]\dfrac{\dot{\psi}}{v} \\ \dfrac{\mu}{v\,\Big(l_r+l_f\Big)}\Big(C_{S,f}\Big(g\, l_r - a_x h_{cg}\Big)\,\delta\ -\ \Big[C_{S,r}\Big(g\, l_f + a_x h_{cg}\Big) \\ +\ C_{S,f}\Big(g\, l_r - a_x h_{cg}\Big)\Big]\,\beta \\ +\ \Big[C_{S,r}\Big(g\, l_f + a_x h_{cg}\Big)\, l_r\ -\ C_{S,f}\Big(g\, l_r - a_x h_{cg}\Big)\, l_f\Big]\dfrac{\dot{\psi}}{v}\Big)\ -\ \dot{\psi} \end{bmatrix}$$

For smaller velocities $|v| < 0.1$, the dynamics simplify to:

$$f(x,u) = \begin{bmatrix} v\cos(\psi+\beta) \\ v\sin(\psi+\beta) \\ v_\delta \\ a_x \\ \dot{\psi} \\ \dfrac{1}{l_{wb}}\left(a_x\ \cos(\beta)\ \tan(\delta)\ -\ v\ \sin(\beta)\ \tan(\delta)\,\dot{x}_7\ +\ \dfrac{v\ \cos(\beta)}{\cos^2(\delta)}\, v_\delta\right) \\ \dfrac{1}{1+\Big(\tan(\delta)\frac{l_r}{l_{wb}}\Big)^2}\cdot\dfrac{l_r}{l_{wb}\ \cos^2(\delta)}\, v_\delta \end{bmatrix}$$

We consider a vehicle of length $l = 4.298\,\mathrm{m}$ and width $w = 1.674\,\mathrm{m}$, with total mass $m = 1.225 \times 10^3\,\mathrm{kg}$ and moment of inertia $I_z = 1.538 \times 10^3\,\mathrm{kg\,m^2}$. The center of gravity is located $l_f = 0.883\,\mathrm{m}$ from the front axle and $l_r = 1.508\,\mathrm{m}$ from the rear axle, at a height $h_{cg} = 0.557\,\mathrm{m}$. The front and rear cornering stiffness coefficients are both $C_{S,f} = C_{S,r} = 20.89\,[1/\mathrm{rad}]$, and the friction coefficient is $\mu = 1.048$.

Replanning Strategy

In our simulation, we employed a replanning strategy to implement a feedback mechanism. In addition to the time horizon, we implemented a fixed time interval, shorter than the time horizon, after which the planner recalculates the trajectory from the current position of the vehicle. This replanning mechanism allows the planner to adjust to changes in the environment or deviations from the planned path. We employed a time discretization of equal intervals for the replanning time interval, and the time intervals increase linearly for the remaining time horizon.

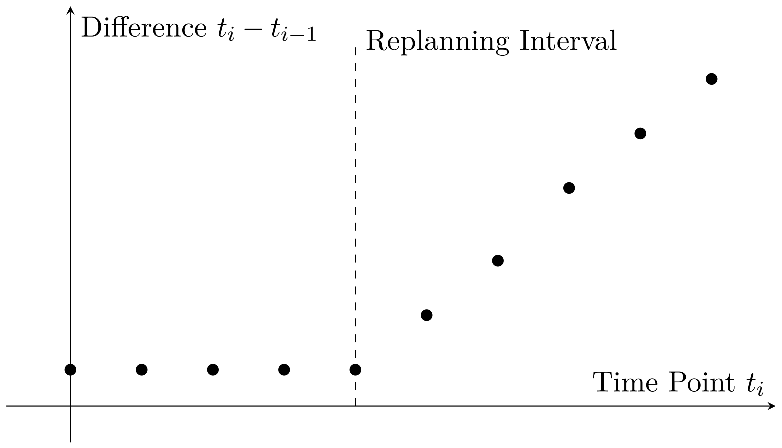


Figure 4.2: Time points and their differences to previous time points

Figure 4.2 illustrates the time points and their intervals. The states for the time points to the left of the dashed line are simulated, after which a replanning is triggered. This replanning follows the same time discretization pattern as the initial planning phase. The term Δt refers to the time intervals between the initial time points, which vary across the different benchmark scenarios. The slope of time difference after the replanning interval is given by $\Delta^2 t_{replan}$.

4.2.4 Results

Point Mass Model

Assumptions and Simplifications. The point mass model simplifies the vehicle to a single mass concentrated at a point. It assumes that the path is primarily driven by the longitudinal and lateral accelerations as constrained by the road alignment. Key kinematic effects such as tire slip angles, lateral load transfer, or steering geometry are either omitted or treated in a highly simplified manner. Furthermore, it often assumes that the orientation of the mass aligns with the road direction, so the model is well suited for higher-level path planning over a known trajectory, where the exact wheel and steering configuration are less critical.

Performance Characteristics. Because of its simpler nature, the point mass model generally offers:

- **Lower computational cost**, enabling faster simulations and easier real-time implementation for basic path tracking.
- **Reduced parameter dependency**, as it needs fewer vehicle parameters (e.g., just mass and approximate friction limits).
- **Reduced accuracy in nonlinear conditions**, because it neglects steering geometry, tire slip angles, and more nuanced lateral dynamics—making it less accurate at high lateral accelerations or large steering angles.

Kinematic Bicycle Model

Assumptions and Structure. The kinematic bicycle model represents the vehicle by a single front tire and rear tire, capturing the geometric relationship among steering angle, vehicle length, and lateral motion. It models the steering input at the front axle while assuming no explicit slip at the tire contact patch (although slip angles may be implicitly captured via kinematic relations). Unlike a full dynamic model, it generally omits complex tire forces and weight transfer, but retains essential geometry required to describe the vehicle's heading and turning radius accurately

Performance Characteristics. Compared to the point mass model, the kinematic bicycle model typically has:

- **Improved representation of vehicle posture**, because it tracks yaw angle and the relation between front and rear axle motions.
- **Better handling of moderate steering maneuvers**, as it can capture how steering inputs alter the effective turning radius and orientation.
- **Moderate computational complexity**, still significantly lighter than full dynamic models but more detailed than a point mass approach.
- **Limitations at high speeds or extreme maneuvers**, as it ignores tire slip forces and load transfers, thus limiting predictive capability in aggressive cornering or low-adhesion conditions.

4.3 Challenges and Limitations

This section discusses the various challenges encountered during the development and implementation of the control and planning algorithms for the vehicle. It highlights specific issues related to replanning, dynamics lag, and the application of McCormick relaxations. Each subsection provides a detailed explanation of the problem, the approach taken to address it, and the results obtained. The purpose of this section is to provide insight into the practical difficulties faced, and the solutions devised to overcome them, thereby contributing to the overall understanding and improvement of the system's performance.

4.3.1 Replanning Challenges for Point Mass

During the replanning phase for the point mass, we encountered a significant challenge. Our control layer introduced some inaccuracies, resulting in the vehicle reaching a higher velocity than anticipated at the time of replanning. This discrepancy led to infeasibility issues during the replanning process.

To address this problem, we introduced soft constraints. These soft constraints allowed for a more flexible approach to handling the velocity overshoot, ensuring that the replanning process could still generate feasible trajectories despite the initial inaccuracies. By incorporating these soft constraints, we were able to mitigate the impact of the velocity overshoot and maintain the overall feasibility of the replanning process.

4.3.2 Lag in Dynamics

We experienced a lag caused by using the forward Euler method on the vehicle model's dynamics and adding a feedback loop to adjust the vehicle orientation. The issue arose because a change in the steering was accounted for in the orientation two additional steps later, leading to overcorrection and resulting in oscillation, illustrated by Figure 4.3.

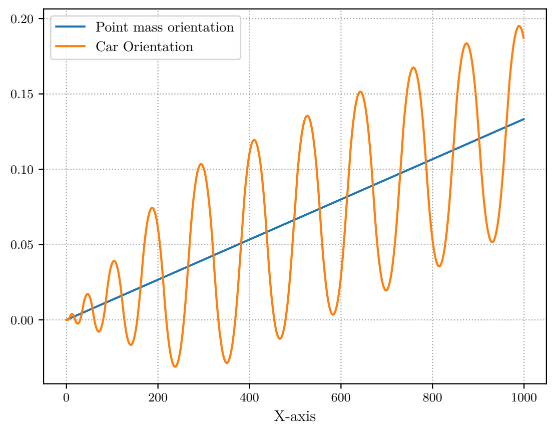


Figure 4.3: Oscillation in vehicle orientation due to lag in dynamics.

To address this problem, we employed a better discretization scheme and accounted for the lag in the feedback loop. By doing so, we were able to mitigate the overcorrection and eliminate the oscillation, leading to a more stable and accurate control of the vehicle's orientation.

4.3.3 McCormick Relaxation

To illustrate the application of these relaxations in practice, consider a path-planning scenario with $v_{min} = 1$, $v_{max} = 5$, and $v_{start} = 1$. In this scenario, the bilinear term $v\xi$, which appears in the equation of motion for $\dot{n} = v \sin \xi \approx v\xi$, is approximated using McCormick relaxations.

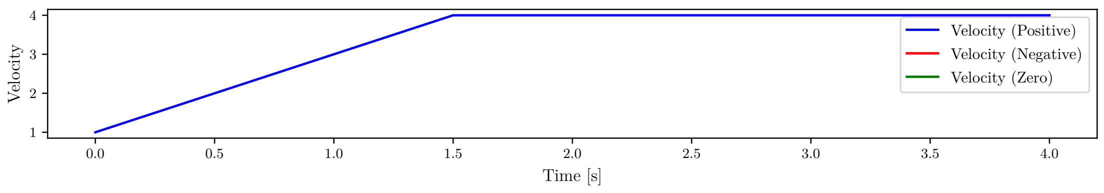


Figure 4.4: Planned velocity profile.

Figure 4.4 shows the planned velocity profile, which quickly reaches its upper limit. The alignment error ξ is close to zero. Here, ξ is bounded within $-45° \leq \xi \leq 45°$. It is noteworthy that ξ does not reach these bounds.

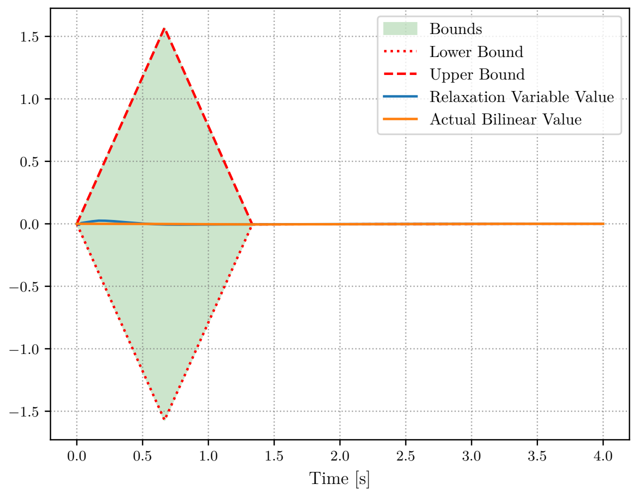
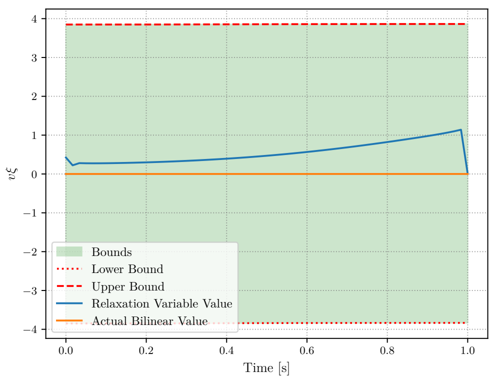


Figure 4.5: McCormick Relaxation on $v\xi$.

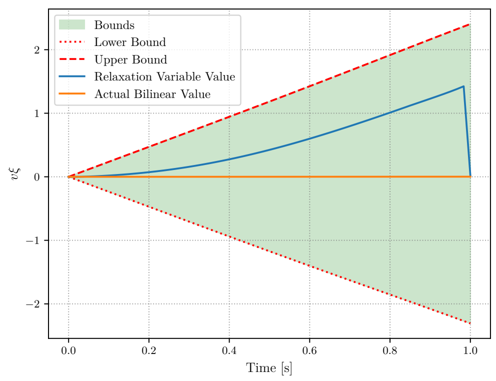
Figure 4.5 compares the actual bilinear value $v\xi$ with the relaxation variable w introduced via McCormick envelopes. This comparison highlights the accuracy of the relaxation approach in approximating the bilinear interaction and its effect on the state transition of n. Once the velocity reaches its limit, the approximation becomes increasingly accurate.

Challenges with McCormick Relaxations

McCormick performs poorly when the velocity is at the midpoint of its limits. Since we decouple the lateral movement from the steering angle and velocity using McCormick relaxations, it is possible for the vehicle to move laterally without any longitudinal movement. To prevent this, we use an approach that allows for tighter bounds on the velocity by considering the maximum and minimum acceleration. Figure 4.6a illustrates the McCormick relaxation without tighter bounds, while Figure 4.6b demonstrates the improvement with tighter bounds.



(a) State transition approximation for n using bilinear term relaxation.



(b) State transition approximation for n using tighter bounds.

Figure 4.6: Comparison of McCormick relaxation with and without tighter bounds.

Due to the rapid inaccuracy of the bounds, more frequent replanning is required. This presents a significant trade-off compared to the point mass model, which necessitates less frequent replanning.

5 Discussion and Future Work

6 Conclusion

List of Figures

1.1 Overview of Autonomous Driving Problem Decomposition 1

2.1 Bicycle model representation of a vehicle. 10
2.2 Frenet Frame Representation 10

3.1 Illustrating the cells with shaded regions. 20
3.2 Illustrating the remaining cells. 20
3.3 In Green our first approach and in Blue using CAD. 21
3.4 Comparison of two sets of intervals for state variables and control inputs. 23
3.5 McCormick relaxation bounds for the bilinear term xy. 29
3.6 McCormick relaxation with tighter bounds on x. 29
3.7 McCormick relaxation constraints for the bilinear terms. 31
3.8 Plot of w versus a and its approximation. 33
3.9 Comparison of the resulting diamond constraints for different acceleration values. 34
3.10 Plots the distance to the road boundaries and the objective function. . . . 36

4.1 Elchtest scenario visualization 41
4.2 Time points and their differences to previous time points 44
4.3 Oscillation in vehicle orientation due to lag in dynamics. 47
4.4 Planned velocity profile. 47
4.5 McCormick Relaxation on $v\xi$. 48
4.6 Comparison of McCormick relaxation with and without tighter bounds. . 49

List of Tables

4.1 Overview of Road Segments and Their Properties 41

Bibliography

[1] J. Eilbrecht and O. Stursberg. “Challenges of Trajectory Planning with Integrator Models on Curved Roads∗.” In: *IFAC-PapersOnLine*. 21st IFAC World Congress 53.2 (Jan. 2020), pp. 15588–15595. ISSN: 2405-8963.

[2] H. K. Khalil. *Nonlinear Systems*. en. Google-Books-ID: t_d1QgAACAAJ. Prentice Hall, 2002. ISBN: 978-0-13-067389-3.

[3] B. F. Caviness, J. R. Johnson, B. Buchberger, and G. E. Collins, eds. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Texts and Monographs in Symbolic Computation. Vienna: Springer, 1998. ISBN: 978-3-211-82794-9 978-3-7091-9459-1.

[4] M. England, R. Bradford, and J. H. Davenport. “Cylindrical algebraic decomposition with equational constraints.” In: *Journal of Symbolic Computation*. Symbolic Computation and Satisfiability Checking 100 (Sept. 2020), pp. 38–71. ISSN: 0747-7171.

[5] M. Werling, J. Ziegler, S. Kammel, and S. Thrun. “Optimal Trajectory Generation for Dynamic Street Scenarios in a Frenet Frame.” In: June 2010, pp. 987–993.