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

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Ich versichere, dass ich diese Bachelor's Thesis selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Munich, 28.02.2025 Andreas Belavic

Abstract

Motion planning is a critical component of robotics and autonomous systems, requiring algorithms that efficiently generate safe and feasible trajectories. This thesis investigates motion planning for autonomous vehicles by framing it as an optimization problem that balances safety, efficiency, and computational feasibility. A central challenge in the optimal control of autonomous driving is the vehicle dynamics model, where a trade-off exists between model accuracy and computational speed. This trade-off becomes especially significant in real-time planning for dynamic environments, where vehicles must navigate while accounting for constraints such as road boundaries, vehicle dynamics, and interactions with other agents.

To address these challenges, we develop optimization-based motion planning approaches using both the double integrator and bicycle models. Our methodology reformulates the motion planning problem within a convex optimization framework, enabling the efficient computation of feasible trajectories while preserving computational tractability. Initially, the approach employs a double integrator model augmented with additional constraints to capture vehicle dynamics without sacrificing convexity. By formulating the problem in the Frenet coordinate frame, we exploit a structured representation that simplifies trajectory generation. System linearization, constraint reformulation, and convex relaxation techniques are applied to manage nonlinearities, ensuring both solution robustness and computational efficiency.

To enhance model fidelity while retaining computational efficiency, the methodology is further extended to incorporate a bicycle model. This extension allows for a more accurate representation of vehicle dynamics, particularly in scenarios demanding realistic steering and acceleration constraints. The bicycle model is approximated to maintain convexity, utilizing techniques such as bilinear term relaxation and auxiliary variables to manage nonconvex components. The proposed approach is evaluated through simulations to assess its performance.

The results demonstrate that the framework can generate feasible and efficient trajectories while maintaining computational efficiency. The evaluation confirms its applicability across a variety of planning scenarios, underscoring its potential for use in autonomous driving applications.

Contents

1 Introduction

1.1 Overview

The motion planning system for autonomous driving is typically organized into a hierarchical, layered architecture that efficiently handles both large-scale navigation and local trajectory optimization. Figure 1.1 illustrates this decomposition, adapted from [1], and emphasizes the motion planning layer, which is the focus of this work.

At the highest level, the Route Planning component takes a user-specified destination and uses search- or graph-based methods to generate a high-level route through the road network. This global planner addresses large-scale navigation by selecting a sequence of waypoints that outline the overall path from the start location to the destination.

Following route planning, the Behavioral Layer refines the generated waypoints by incorporating environmental factors such as other vehicles, obstacles, and road signs. This step ensures that the vehicle adapts to dynamic traffic conditions and selects a behavioral strategy appropriate for the current driving context.

Once the behavioral strategy is set, the Motion Planning Layer takes center stage. Building on insights from early demonstrations in the DARPA Grand and Urban Challenges [2, 3], the motion planning literature has evolved to tackle a wide variety of environments—from structured highways and urban roads to unstructured parking lots and off-road terrains. A unifying theme across these scenarios is the production of collision-free, dynamically feasible trajectories that account for comfort and operational constraints.

In this context, an optimization-based local planner refines the global route into a smooth, continuous guiding trajectory in real time [4]. Unlike the global route—represented as a series of discrete waypoints—this trajectory not only satisfies road rules and safety constraints but also ensures fluid, comfortable motion that adapts to dynamic obstacles,

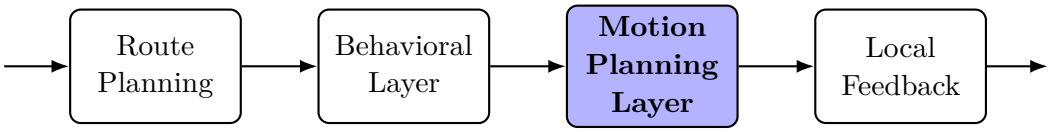


Figure 1.1: Overview of Motion Planning Problem Decomposition

road curvature, and other real-time factors.

Another critical element in the motion planning pipeline is the vehicle model. This model defines the vehicle's position and orientation in the real world and predicts how these states evolve over time. The choice of vehicle model significantly impacts both the accuracy and the computational complexity of the planning process: simpler models offer efficiency at the cost of precision, whereas more complex models yield higher fidelity with increased computational demands.

Finally, the Local Feedback component executes the planned trajectory by generating precise control commands—steering, throttle, and brake inputs—based on real-time vehicle and environmental feedback. This closed-loop control ensures that the trajectory is followed accurately, even as conditions evolve.

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 while ensuring collision avoidance and adherence to vehicle dynamics. Compared to graph-based or Incremental Tree approaches, optimization-based methods provide smoother and more dynamically feasible trajectories by directly incorporating vehicle dynamics constraints. Additionally, they offer deterministic performance with guaranteed feasibility, avoiding the sampling inconsistencies and discontinuities often found in tree-based methods. By leveraging convex optimization and constraint reformulation, this approach efficiently computes safe and feasible trajectories while maintaining real-time performance in dynamic environments.

Building on these advantages, we review existing optimization-based motion planning approaches, highlighting key methodologies, challenges, and advancements in the field.

1.2 Related Work

Several surveys have synthesized the state of the art in motion planning for autonomous driving. Claussmann et al. compare classical graph-search algorithms, sampling-based methods, and optimization-based planners, emphasizing the need for trajectories that respect vehicle dynamics and safety constraints, particularly in high-speed highway scenarios [5]. They highlight the role of Model Predictive Control (MPC) in generating feasible and efficient trajectories.

For urban driving, Paden et al. categorize planning methods into variational, graph-based, and sampling-based approaches, stressing how model fidelity and computational constraints influence method selection [1]. They highlight the effectiveness of convex formulations, such as quadratic programming, in making trajectory optimization tractable while balancing accuracy and efficiency.

González et al. provide a broader review, identifying two key challenges: avoiding moving obstacles and ensuring real-time re-planning in dynamic environments [6]. They note advancements in computing and optimization that enable real-time trajectory generation but emphasize the trade-off between solution optimality and computational feasibility.

Across these works, a common theme emerges: motion planning must balance vehicle dynamics, safety constraints, and computational efficiency. Convex optimization has become a crucial tool in achieving this balance, setting the stage for the methods explored in this thesis.

1.2.1 Classical Motion Planning Approaches

Early motion planning research focused on finding feasible paths in a geometric sense, often treating the vehicle as a point mass and ignoring detailed dynamics. Graph-based algorithms (e.g., A*, D*) discretize the state or configuration space into grids or lattices to search for collision-free paths, but they can become computationally intractable in high-dimensional spaces. Sampling-based planners, such as Probabilistic Roadmaps (PRM) and Rapidly-exploring Random Trees (RRT), offered a breakthrough in efficiency by randomly sampling the continuous configuration space [7]. PRM methods construct a graph of randomly-sampled collision-free configurations, while RRT incrementally grows a tree from the start state toward the goal by sampling random inputs [7]. These methods are probabilistically complete, meaning they will find a solution if one exists given enough time. Variants like RRT* and PRM* were later developed to achieve asymptotic optimality (convergence towards the optimal path as samples increase) for metrics like path length.

Despite their success in finding feasible paths even in high-dimensional or complex

environments, classical planners have notable limitations. They typically produce geometric paths without timing information, requiring a separate velocity profile or trajectory generation step to account for kinematics and dynamics. Moreover, they do not inherently optimize a specific cost function (beyond maybe path length for RRT*), so the resulting paths may be suboptimal in terms of travel time, energy, or comfort. In practice, an initial path from a sampling or graph search planner often needs refinement. For example, trajectory optimization algorithms are commonly employed to smooth and shorten a raw path from an RRT or grid planner [8], adjusting the path points or timing to satisfy dynamic constraints and improve optimality. This two-stage approach (planning then post-optimization) can work, but it separates feasibility and optimality into distinct steps. The transition to optimization-based methods aims to unify these steps by directly computing trajectories that are both feasible and optimal under the given criteria.

1.2.2 Optimization-Based Motion Planning

Rather than searching discretely for a path, trajectory optimization treats motion planning as a continuous optimization problem: finding a time-parametrized state/control sequence that minimizes a cost (such as time or energy) while satisfying system dynamics and constraints. This approach is rooted in optimal control theory. Modern optimal control solvers often use direct methods (also known as transcription methods) which discretize the continuous-time problem into a finite-dimensional problem. In a direct transcription approach, the time horizon is divided into N intervals, and the state x_n and control input u_n at each time step are treated as decision variables. The vehicle's discrete-time dynamics $x_{n+1} = f(x_n, u_n)$ are imposed as constraints linking these variables across time [9]. This transforms the trajectory planning task into a large constrained optimization (typically a nonlinear program). When the system dynamics are linear (or linearized) and constraints and costs are convex (e.g., quadratic cost, linear constraints), the transcription yields a convex optimization problem – specifically a linear or quadratic program that can be solved efficiently to global optimality [9]. In such cases, algorithms can reliably find the optimal trajectory in real time, even online within a model predictive control loop [9].

However, for a car-like vehicle, the true dynamics are nonlinear and involve non-convex constraints (especially for obstacle avoidance). Directly transcribed optimal trajectory planning generally leads to a non-convex problem that standard solvers can only solve to local optimality. A rich body of work has explored strategies to cope with this non-convexity. One strategy is to exploit sequential convex optimization: the idea is to iteratively approximate the non-convex problem with a sequence of convex problems that are easier to solve. For instance, Schulman et al. (2014) present a trajectory optimizer that at each iteration linearizes or convexifies the collision avoidance constraints and

solves a convex sub-problem; by gradually tightening the approximation, the method converges to a feasible local optimum [8]. Such sequential convex programming approaches (including algorithms like CHOMP and TrajOpt) have demonstrated fast convergence on complex motion planning tasks, effectively handling high-dimensional robots by solving a series of convex sub-problems. The key advantage is that each convexified problem can be solved to global optimality, providing a well-defined improvement at each iteration. The downside is that global optimality of the overall non-convex problem is not guaranteed – the result depends on the quality of the convex approximations and initial guesses. Nevertheless, these methods illustrate the central role of convex optimization in modern motion planning: even when the full problem is non-convex, formulating as much as possible in convex terms (dynamics, cost, safe regions, etc.) leads to more robust and efficient solvers.

Another prominent trajectory optimization framework is Model Predictive Control (MPC) [10, 11], which can be seen as a particular implementation of optimal control solved repeatedly in a receding horizon. In MPC, at each time step a finite-horizon optimal control problem is solved (often by direct transcription as above) to yield an optimal trajectory and control sequence, of which only the first control action is applied before the horizon slides forward. MPC has been widely adopted in vehicle trajectory planning and tracking due to its ability to handle multi-constraint, multi-variable control problems in real time. To ensure tractability, MPC formulations for vehicles often make simplifying assumptions on the vehicle model. Depending on the scenario, MPC can employ either a kinematic model or a dynamic model, and either use the full nonlinear model or a linearized version of it [12]. Nonlinear MPC (using the full vehicle dynamics) provides higher fidelity and can accurately capture system behavior, but at the cost of significant computational load [12]. For real-time performance, especially in fast update cycles, practitioners commonly use linear time-varying (LTV) MPC or linear time-invariant (LTI) MPC, where the vehicle model is linearized around a current operating point or along a reference trajectory [12]. This yields a convex quadratic program at each MPC step, greatly reducing computation time while still accounting for vehicle constraints (like acceleration limits, steering limits, etc.) [12]. However, even with linearization, solving constrained optimal control problems in real time remains computationally demanding, especially at high planning frequencies. To further enhance efficiency, many trajectory planners adopt simplified motion models that maintain convexity while retaining key dynamic properties. One such approach is the use of linear integrator models, which abstract vehicle motion into a set of linear equations, enabling fast convex optimization.

1.2.3 Vehicle Dynamics Modeling in Planning

Linear Integrator Models for Convex Optimization

Linear integrator models (e.g. double or triple integrators for position, velocity, and acceleration) are popular in trajectory planning because they yield linear dynamics that make the optimization problem convex. With such models, vehicle dynamics are expressed as simple linear equations (position as a double-integrator of acceleration, etc.) . For instance, modeling motion with a triple integrator (including acceleration as a state and jerk as control) lets one impose acceleration limits as linear state constraints [13]. Esterle et al. leverage this property to encode vehicle non-holonomic behavior (orientation) and collision avoidance with linear constraints, using a disjunctive (mixed-integer) formulation to approximate the otherwise nonlinear orientation constraints [13]. By approximating non-holonomy in this way, they maintain a convex formulation (MIQP) for the planner while still respecting vehicle orientation limits, and show that the resulting trajectories remain feasible when validated with a full nonlinear model [13]. This demonstrates how integrator-based models can incorporate complex vehicle constraints through clever linear approximations, preserving convexity of the optimization.

However, a challenge arises when planning on curved roads or general road topologies using simple integrator models. In a global frame, ensuring the vehicle stays within lane/road boundaries introduces coupling between the vehicle's lateral and longitudinal motion (e.g. curvature and speed constraints) that is generally nonlinear. Eilbrecht and Stursberg point out that planning with linear integrator dynamics on a curved road inherently produces non-convex coupled constraints on states and inputs [14]. For example, limiting lateral acceleration or enforcing a maximum curvature for a given speed couples the longitudinal velocity and lateral position in a non-convex way. This coupling can impede computationally efficient planning [14]. To address this, they propose modifying such constraints via inner approximations – effectively relaxing or convexifying the road-curvature constraints [14]. By computing convex inner approximations of the feasible region (using methods like quantifier elimination), they transform the originally non-convex road constraints into conservative convex constraints [14]. The result is that the planner can safely account for road geometry (staying in lane on curved roads) without losing the convex nature of the problem. This insight underlines the need for carefully formulated constraints: even with a double-integrator model, one often must introduce additional constraints (like lateral acceleration limits, curvature bounds, or road boundary conditions) and approximate them in convex form to retain a tractable optimization problem.

Kinematic vs. Dynamic Bicycle Models and Constraints

Vehicle models range from simple kinematic bicycles (single-track models without tire slip dynamics) to more complex dynamic models that include tire forces and slip. A key trade-off is that kinematic models are simpler and can often be handled in (or approximated to) a convex optimization framework, whereas dynamic models are more accurate but lead to non-convex formulations that usually require nonlinear or mixed-integer programming. Kong et al. compare these two model types and highlight this trade-off [15]. In their study, a kinematic single-track model (bicycle model) was used in an MPC controller and contrasted with a full dynamic single-track (tire-force) model. They found that the kinematic model can achieve similar prediction accuracy as the dynamic model when discretized appropriately, while being significantly less computationally expensive [15]. Notably, using a coarse update rate (200 ms) with the kinematic model yielded accuracy comparable to a 100 ms update with the dynamic model [15]. At identical update rates, the kinematic model actually predicted vehicle motion more accurately than the dynamic model in their tests [15]. This indicates that for many planning scenarios (especially at moderate speeds), a well-tuned kinematic model is sufficient and can be used with a larger time-step, directly reducing computation load.

Another advantage of the kinematic bicycle model noted by Kong et al. is its numerical robustness at low speeds. The dynamic model with tire equations can suffer singularities or poor conditioning as speed approaches zero (e.g. in stop-and-go scenarios), whereas the kinematic model has no such issue [15]. This means a planner based on a kinematic model can naturally handle scenarios like stop-and-go traffic or tight maneuvers, which might complicate a dynamic model-based optimizer. Indeed, Kong et al. report that their kinematic-model MPC could handle a wide range of speeds (including zero) with lower computational cost, while the dynamic model-based controller struggled in very low-speed regimes [15]. The flip side is that at higher speeds (where significant lateral tire forces and slip arise), the dynamic model outperforms the kinematic model in accuracy [15]. In other words, the kinematic model is convex-friendly and adequate for low-to-mid speed planning, but one must impose additional constraints to ensure it stays within its valid regime; beyond that regime (e.g. high-speed cornering), a dynamic model (which leads to a non-convex problem) would capture necessary effects like tire slip more accurately [15].

To reconcile the use of a simpler model with vehicle limits, researchers introduce constraints or adaptive limits into kinematic model planners. One common approach is to enforce a speed-dependent curvature or steering constraint to mimic the handling limits of the vehicle. Polack et al. implement this by limiting the steering angle as a function of forward velocity in their kinematic bicycle MPC planner [16]. This effectively imposes an upper bound on curvature (or lateral acceleration) at high speeds – a direct analog of an "adaptive lateral speed constraint". By doing so, they ensure the planned trajectory

remains feasible for the real vehicle at all times [16]. In fact, Polack et al. emphasize that this speed-conditioned steering limit "ensures the validity of the kinematic bicycle model at any time" [16]. In practice, this means the planner will automatically slow down the vehicle (or reduce steering commands) for sharp turns, respecting a friction or handling limit without needing a full dynamic model. The steering-angle constraint coupled with speed is a nonlinear coupling (since allowable steering depends on velocity), but it can be handled by conservative approximation or by embedding it as a lookup constraint in the MPC. This kind of model-based constraint is an example of a relaxation technique: it keeps the optimization mostly convex (the kinematic model and other constraints remain linear or convex), by carving out any solutions that would violate the vehicle's true dynamic limits. Similarly, Polack's two-level architecture [16] uses a 10 Hz kinematic-MPC planner with these constraints, then a 100 Hz low-level controller to track the plan. This ensured consistency between the planned path and what the dynamic controller could actually follow, effectively guaranteeing that the convex planner's output was feasible for the real car's dynamics.

In summary, kinematic bicycle models with added constraints combine the best of both worlds: they retain the convex (or easily linearized) structure of the motion planning problem while capturing essential vehicle limitations. If a planner uses a pure double-integrator or kinematic model without such constraints, it may generate trajectories that a real vehicle cannot follow (e.g. demanding too high lateral acceleration on a curve). The cited works show how to avoid that: by integrating road geometry and vehicle handling constraints (e.g. lateral acceleration, curvature, or steering limits that depend on speed) into the model, one can maintain a convex optimization formulation that is both computationally efficient and yields physically feasible trajectories [16, 14].

1.2.4 Summary

The reviewed literature highlights the critical trade-offs in motion planning: simplified models enable convex optimization and real-time feasibility, but additional constraints must be imposed to ensure trajectory feasibility. Linear integrator models provide a computationally efficient formulation but require convex approximations for road geometry and vehicle constraints. Kinematic models, when constrained appropriately, offer a balance between efficiency and dynamic feasibility, while dynamic models provide higher accuracy at a significant computational cost.

Building upon these insights, this thesis develops a motion planning framework that leverages convex formulations of vehicle dynamics and road topology constraints. The next section formalizes this problem statement.

1.3 Problem Formulation

Motion planning for autonomous vehicles must simultaneously address two fundamental challenges: ensuring collision-free trajectories and accurately capturing the vehicle's dynamic behavior. In this section, we introduce a discrete-time formulation of the trajectory planning problem and discuss its inherent computational complexity. This lays the groundwork for our convex optimization approach.

1.3.1 Computational Complexity

Optimal trajectory planning is computationally intractable and classified as PSPACE-Hard [17], with resource requirements growing exponentially with problem size. This complexity arises from the continuous state space, non-convex vehicle dynamics constraints, and the need to enforce collision avoidance while satisfying dynamic constraints. These challenges, akin to the classic Movers' Problem, are further complicated by time dependencies and evolving vehicle behavior. Exact solutions are impractical for real-time use, necessitating numerical optimization, heuristics, or approximate solvers for efficient near-optimal solutions.

1.3.2 Discrete-Time Problem Formulation

To render the trajectory planning problem tractable, we discretize the continuous-time formulation 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. We define the trajectory at discrete time points $\{t_i\}_{i=1,\dots,m}$, with the state at time t_i denoted by $\pi(t_i) = x_i$. Our objective is to minimize a cumulative cost function $J : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$, which penalizes deviations from desired behaviors, including collisions and dynamic infeasibilities.

Discrete-Time Optimal Trajectory Planning

Given a 7-tuple $(\mathcal{X}, \mathcal{U}, x_{\text{initial}}, X_{\text{goal}}, f, J, \{t_i\}_{i=1,\dots,m})$, the discrete-time optimal trajectory planning problem is defined as:

$$u^* = \underset{u \in \mathcal{U}^{m-1}}{\arg\min} \sum_{i=1}^{m-1} J(x_{i+1}, u_i), \tag{1.1}$$

$$\text{s.t.} \quad x_1 = x_{\text{initial}} \tag{1.2}$$

$$x_m \in X_{\text{goal}} \subseteq \mathcal{X} \tag{1.3}$$

$$(x_i, u_i) \in \mathcal{C} \subseteq \mathcal{X} \times \mathcal{U} \qquad \text{for all } i \in \{1, \dots, m-1\} \tag{1.4}$$

$$x_{i+1} = x_i + \Delta t_i \, f(x_i, u_i) \qquad \text{for all } i \in \{1, \dots, m-1\} \tag{1.5}$$

where $\Delta t_i = t_{i+1} - t_i$, and $\mathcal{C}$ represents the set of coupling constraints that jointly enforce collision avoidance and the vehicle's dynamic limitations. The system dynamics are approximated using an integration scheme, as specified in (1.5), which estimates the state at time t_{i+1} 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 over the feasible set 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 bounded above by 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.

1.4 Thesis Structure

Chapter 2 introduces the vehicle modeling approaches such as the double integrator 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

This section introduces two fundamental models used in trajectory planning: the double integrator model, which offers a simplified kinematic representation, and the kinematic bicycle model, which captures essential vehicle dynamics. Both models are detailed in [18]. 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:

$$\dot{x} = f(x, u) \tag{2.1}$$

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

2.1 Double Integrator Model

The double integrator model (DI) 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:

$$x = \begin{bmatrix} p_x & p_y & v_x & v_y \end{bmatrix}^T \tag{2.2}$$

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:

$$u = \begin{bmatrix} a_x & a_y \end{bmatrix}^T \tag{2.3}$$

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

$$\dot{x} = \begin{bmatrix} v_x & v_y & a_x & a_y \end{bmatrix}^T \tag{2.4}$$

Additionally, the control inputs are constrained by:

$$\sqrt{u_1^2 + u_2^2} \leq a_{\max} \tag{2.5}$$

where a_{max} limits the maximum allowable acceleration. Despite its simplicity, the double integrator 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 Kinematic Bicycle Model

The kinematic 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:

$$x = \begin{bmatrix} p_x & p_y & \delta & v & \psi \end{bmatrix}^T \tag{2.6}$$

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:

$$u = \begin{bmatrix} v_\delta & a \end{bmatrix}^T \tag{2.7}$$

where v_δ is the steering velocity, and a is the longitudinal acceleration.

The vehicle's motion follows the kinematic equations:

$$\dot{p}_x = v\cos(\psi) \tag{2.8}$$
$$\dot{p}_y = v\sin(\psi) \tag{2.9}$$
$$\dot{\delta} = v_\delta \tag{2.10}$$
$$\dot{v} = a \tag{2.11}$$
$$\dot{\psi} = \frac{v}{l_{wb}}\tan(\delta) \tag{2.12}$$

where l_{wb} 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.

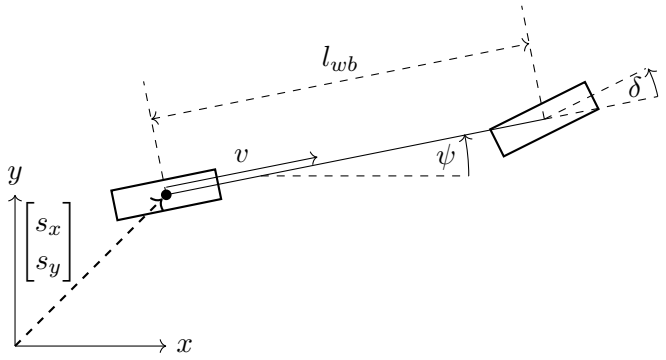


Figure 2.1: Bicycle model representation of a vehicle.

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

$$\sqrt{a^2 + (v\dot{\psi}_5)^2} \leq a_{\max} \tag{2.13}$$

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.

2.3 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.

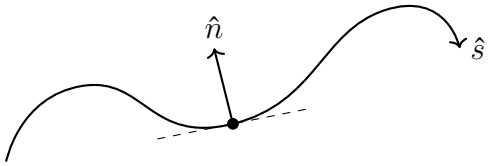


Figure 2.2: Frenet Frame Representation

3 Methodology

3.1 Motion Planning Using the Double Integrator Model

This section develops a motion planning framework using the double integrator model from 2.1, transitioning from a global coordinate representation to a Frenet frame formulation. The Frenet frame provides a more structured approach for path tracking along a predefined road, accounting for curvature and alignment errors. Our formulation builds on the work of Eilbrecht et al. [14] and systematically addresses motion dynamics, constraints, and control strategies.

3.1.1 Overview of the Motion Planning Framework

To effectively model and control vehicle motion, we employ a structured approach that systematically integrates kinematics, dynamics, constraints, and control transformations.

We begin by defining the **coordinate transformation and kinematics**, introducing the curvature $C(s)$ of the reference path and the alignment error $\xi = \psi - \theta$. These definitions allow us to derive the first- and second-order kinematic equations, which describe how the vehicle's body-fixed velocity components influence its evolution in Frenet coordinates.

Next, we formulate the **system dynamics** within the Frenet frame, defining the state vector and control inputs. By incorporating the effects of curvature and alignment error, we capture both longitudinal and lateral dynamics within our model.

To address the nonlinearities introduced by the curvature terms, we employ **feedback linearization and system linearization**. This process involves assuming alignment to simplify the nonlinear dynamics, enabling the introduction of artificial control variables that linearize the system.

To ensure that the planned trajectories remain within physical and operational limits, we carefully handle **constraint formulation**. We derive bounds on velocity and acceleration and map them from the body-fixed frame to the Frenet frame, thereby enforcing vehicle and road constraints within the planning model.

A key challenge in motion planning is the presence of **non-convexities** introduced by curvature-dependent constraints. To address this, we use **quantifier elimination** techniques to obtain convex inner approximations of the feasible set. We explore two

approaches:

- *Interval Fitting*, which provides a computationally efficient, box-shaped approximation of the constraint set.
- *Cylindrical Algebraic Decomposition (CAD)*, a method from computer algebra that decomposes space into cylindrical cells to eliminate quantifiers while preserving logical equivalence.

Once an optimal trajectory is determined, we establish a **control transformation** that maps the optimized state and control variables to physical vehicle inputs. This step derives the necessary steering angle and longitudinal acceleration, ensuring compatibility with standard vehicle controllers.

Next, we provide the **exact discretization of the double integrator model**, ensuring an accurate transition from continuous to discrete-time dynamics. By leveraging matrix exponentials, this formulation preserves system behavior over discrete time steps and maintains numerical stability, which is crucial for real-time motion planning.

Finally, we present the **complete motion planning model** as a formalized representation. This encapsulates the system's state-space dynamics, control inputs, constraints, and transformation mappings.

3.1.2 Coordinate Transformation and Kinematics

In this subsection, we establish the foundation for our motion planning framework by describing the geometry of the reference path and deriving the vehicle's kinematic equations in the Frenet frame.

First, let $\theta(s)$ denote the tangent angle at an arc length s along the reference path. The curvature, $C(s)$, quantifies the rate of change of this tangent angle with respect to s and is defined as:

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

Next, consider the vehicle's orientation, ψ. To measure how much the vehicle deviates from following the road's direction, we define the *alignment error* ξ as:

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

This error quantifies the difference between the vehicle's heading and the path's tangent direction.

Using these definitions and standard coordinate transformation techniques [14], we can derive the vehicle's motion dynamics in the Frenet frame. In this framework, the velocity components in the vehicle's body-fixed frame are directly related to the time derivatives of the Frenet coordinates.

First-Order Kinematics The following equations describe how the vehicle's position evolves along the path:

$$\dot{s}\,(1 - n\,C(s)) = v_x \cos\xi - v_y \sin\xi, \tag{3.3}$$
$$\dot{n} = v_x \sin\xi + v_y \cos\xi, \tag{3.4}$$

where:

- s is the longitudinal position along the reference path,
- n represents the lateral deviation from the path,
- v_x and v_y are the velocity components in the body-fixed frame.

Note that the term $1 - n\,C(s)$ adjusts the longitudinal progress to account for the path's curvature.

Acceleration Dynamics To capture the dynamics of acceleration in the Frenet frame, we introduce transformed acceleration components $a_{x,tn}$ and $a_{y,tn}$. These are defined by:

$$a_{x,tn} = (a_x - v_y\,\dot{\psi})\cos\xi - (a_y + v_x\,\dot{\psi})\sin\xi, \tag{3.5}$$
$$a_{y,tn} = (a_x - v_y\,\dot{\psi})\sin\xi + (a_y + v_x\,\dot{\psi})\cos\xi, \tag{3.6}$$

with the following detailed definitions:

$$a_{x,tn} := \ddot{s}\,(1 - n\,C(s)) - 2\dot{n}\,C(s)\dot{s} - n\,C'(s)\dot{s}^2, \tag{3.7}$$
$$a_{y,tn} := \ddot{n} + C(s)\dot{s}^2\,(1 - n\,C(s)). \tag{3.8}$$

These equations illustrate how the vehicle's acceleration in the Frenet frame is influenced by both its inherent dynamics (through $\ddot{s}$ and $\ddot{n}$) and the geometry of the reference path (through $C(s)$ and its derivative $C'(s)$).

In summary, this subsection defines the key geometric parameters and derives the kinematic equations necessary for representing vehicle motion in the Frenet frame. These results set the stage for the subsequent development of the full motion planning model.

3.1.3 System Dynamics Formulation

We can now formalize the motion planning model in the Frenet frame by defining the state and control input vectors. The state vector, denoted by x_{di}, captures the vehicle's

position, alignment error, and their corresponding velocities:

$$x_{di} = \begin{bmatrix} s \\ n \\ \xi \\ \dot{s} \\ \dot{n} \\ \dot{\psi} \end{bmatrix}, \tag{3.9}$$

where:

- s is the longitudinal position along the reference path,
- n is the lateral deviation,
- ξ is the alignment error $(\psi - \theta)$,
- $\dot{s}$, $\dot{n}$, and $\dot{\psi}$ are the corresponding time derivatives.

The control inputs are represented by the vector u_{di}, which comprises the body-fixed accelerations:

$$u_{di} = \begin{bmatrix} a_x \\ a_y \\ a_\psi \end{bmatrix}. \tag{3.10}$$

Using the kinematic relationships derived earlier in First-Order Kinematics and Acceleration Dynamics, the dynamics of the double integrator model in the Frenet frame can be expressed as:

$$f_{di}(x_{di}, u_{di}) = \begin{bmatrix} \dot{s} \\ \dot{n} \\ \dot{\psi} - C(s)\dot{s} \\ \dfrac{a_{x,tn} + 2\dot{n}\,C(s)\dot{s} + n\,C'(s)\dot{s}^2}{1 - n\,C(s)} \\ a_{y,tn} - C(s)\dot{s}^2(1 - n\,C(s)) \\ a_\psi \end{bmatrix}. \tag{3.11}$$

This formulation captures the vehicle's motion along a curved path by incorporating both its intrinsic dynamics and the influence of road curvature. It is important to note that the curvature terms $C(s)$ and $C'(s)$ introduce non-convexities into the system dynamics. We address these challenges in subsequent sections through feedback linearization.

3.1.4 Linearization via Feedback Control

In order to simplify the nonlinear dynamics introduced by the curvature terms, we first decouple the input by examining the dynamics in (3.11) (with the definitions in (3.7) and (3.8)). Notice that the last two entries contain both body-fixed control inputs a_x and a_y. To simplify the model, we make the following key assumption.

Assumption: Alignment Error

We assume that the vehicle is always aligned with the road, i.e.,

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

This assumption leads to several useful simplifications:

- The body-fixed accelerations are directly given by the transformed accelerations:

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

- The rate of change of the vehicle's orientation becomes:

$$\dot{\psi} = \dot{\theta} = C(s)\,\dot{s},$$

  where $C(s) = \dfrac{d\theta}{ds}$.

- The yaw acceleration satisfies:

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

Although this assumption fixes the orientation to the road, the vehicle is still permitted lateral movement via lateral acceleration.

With this alignment assumption, the dynamics become affine in $a_{x,tn}$ and $a_{y,tn}$. This property allows us to introduce artificial control inputs that will fully linearize the system.

Introducing Artificial Control Inputs

Define the artificial control input vector $\tilde{u}_{di}$ as:

$$\tilde{u}_{di} := \begin{bmatrix} u_t \\ u_n \end{bmatrix} := \begin{bmatrix} \dfrac{a_{x,tn} + 2\dot{n}\,C(s)\dot{s} + n\,C'(s)\dot{s}^2}{1 - n\,C(s)} \\ a_{y,tn} - C(s)\dot{s}^2(1 - n\,C(s)) \end{bmatrix}. \tag{3.13}$$

These artificial control inputs will be used to linearize the system dynamics, a process known as feedback linearization. This technique is thoroughly explained in [19]. In the following section, we will briefly introduce the concept of feedback linearization before delving into its application in our motion planning framework.

Supplementary Background: 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 state-feedback law. Consider a general nonlinear system of the form:

$$\dot{x} = f(x) + G(x)\, u, \tag{3.14}$$

where

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

For feedback linearization, it is typical to assume that the system is fully actuated, i.e.,

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

ensuring that $G(x)$ is invertible in the region of interest. Under this assumption, one can define a new control input v such that:

$$u = G(x)^{-1}\Big[v - f(x)\Big]. \tag{3.15}$$

By canceling the nonlinear dynamics $f(x)$, the new input v governs an equivalent linear system that can be addressed with standard linear control techniques.

Resulting Simplified Model

Returning to our model, we can use the artificial control inputs $\tilde{u}_{di}$ to linearize the system dynamics. Given our assumption of no alignment error 3.1.4, the state variables can be simplified by removing the orientation ψ, as it is fixed to the road. The reduced state vector is:

$$\tilde{x}_{di} = \Big[s, \quad n, \quad \dot{s}, \quad \dot{n}\Big]^T,$$

and the new artificial control inputs are:

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

Under these definitions, the system dynamics simplify to:

$$\tilde{f}_{di}(\tilde{x}_{di}, \tilde{u}_{di}) = \begin{bmatrix} \dot{s} \\ \dot{n} \\ u_t \\ u_n \end{bmatrix}. \tag{3.16}$$

With the dynamics now expressed in a simplified, linear form, our next task is to ensure that the planned trajectories adhere to both the vehicle's physical limitations and the geometric constraints of the road. In the following section, we establish the coupling constraints between the state variables and control inputs for our discrete-time optimal trajectory planning problem (see (1.4)) based on [14].

3.1.5 Constraint Handling

This subsection addresses the constraints imposed by the vehicle's physical limits and the road geometry. We first define the constraints on state variables and control inputs in the body-fixed frame and then map them to the Frenet frame. This mapping allows us to formulate the overall coupling constraint set that must be satisfied during trajectory planning. We will also discuss the challenge of non-convexity arising from these coupling constraints and formulate the problem of finding a convex under-approximation.

Let $\square$ denote any planning variable. For each variable, we define constant upper and lower bounds during planning, denoted by $\overline{\square}$ and $\underline{\square}$, respectively. For example, in the vehicle's body-fixed frame the velocity constraints are expressed as:

$$\underline{v_x} \leq v_x \leq \overline{v_x}, \tag{3.17}$$

$$\underline{v_y} \leq v_y \leq \overline{v_y}. \tag{3.18}$$

By applying the first-order kinematics from (3.3) and (3.4) with the alignment error set to $\xi = 0$, these velocity bounds translate into the Frenet frame as follows:

$$\underline{v_x} \leq \dot{s}\,(1 - n\,C(s)) \leq \overline{v_x}, \tag{3.19}$$

$$\underline{v_y} \leq \dot{n} \leq \overline{v_y}. \tag{3.20}$$

Here, $\dot{s}$ represents the longitudinal speed adjusted by the term $(1 - n\,C(s))$ to account for the curvature of the road, and $\dot{n}$ is the lateral speed.

For acceleration constraints, we typically define two types:

- A *norm constraint* that limits the overall acceleration, ensuring that the combined longitudinal and lateral accelerations lie within a circle of radius c:

$$a_x^2 + a_y^2 \leq c. \tag{3.21}$$

- *Individual bounds* on the longitudinal and lateral accelerations:

$$\underline{a_x} \leq a_x \leq \overline{a_x}, \tag{3.22}$$
$$\underline{a_y} \leq a_y \leq \overline{a_y}. \tag{3.23}$$

To map these acceleration constraints to the Frenet frame, we use the definition of our artificial variables (3.13) and the first implication $a_x = a_{x,tn}$ and $a_y = a_{y,tn}$ from our alignment error assumption 3.1.4. These equations allow us to establish a mapping that relates the state variables and artificial control inputs $(\tilde{x}_{di}, \tilde{u}_{di})$ to the body-fixed accelerations by solving for a_x and a_y. This mapping is defined as:

$$g(\tilde{x}_{di}, \tilde{u}_{di}) := \begin{bmatrix} (1 - n\,C(s))\,u_t - \left(2\dot{n}\,C(s)\dot{s} + n\,C'(s)\dot{s}^2\right) \\ u_n + C(s)\,\dot{s}^2\,(1 - n\,C(s)) \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \end{bmatrix}. \tag{3.24}$$

Substituting this mapping into the individual acceleration constraints, we derive the following bounds in the Frenet frame:

$$\begin{bmatrix} \underline{a_x} \\ \underline{a_y} \end{bmatrix} \leq g(\tilde{x}_{di}, \tilde{u}_{di}) \leq \begin{bmatrix} \overline{a_x} \\ \overline{a_y} \end{bmatrix} \tag{3.25}$$

$$\|g(\tilde{x}_{di}, \tilde{u}_{di})\|^2 \leq c. \tag{3.26}$$

Next, we need to impose limits on the yaw rate and yaw acceleration. The yaw rate is defined as $C(s)\,\dot{s}$, and the yaw acceleration is given by $C'(s)\,\dot{s}^2 + C(s)\,u_t$, as derived from the second and third implications of our alignment error assumption 3.1.4.

$$\underline{\dot{\psi}} \leq C(s)\,\dot{s} \leq \overline{\dot{\psi}}, \tag{3.27}$$
$$\underline{a_\psi} \leq C'(s)\,\dot{s}^2 + C(s)\,u_t \leq \overline{a_\psi}. \tag{3.28}$$

Thus, we have successfully modeled the physical limits of the vehicle in the Frenet frame. The constraints arising from the road topology can be represented using the Frenet frame coordinates from our state variables, s and n. To adhere to DCP rules, we allow the lateral range to depend on the arc length s. This is achieved by defining the bounds $\underline{n}(s)$ and $\overline{n}(s)$ for the lateral deviation, where $\underline{n}(s)$ is convex in s and $\overline{n}(s)$ is concave in s. The arc length s is constrained by the constant bounds $\underline{s}$ and $\overline{s}$.

Combining all introduced constants, the overall coupling constraint set $\mathcal{C}$ is defined as:

$$\mathcal{C} := \left\{ \begin{bmatrix} \tilde{x}_{di} \\ \tilde{u}_{di} \end{bmatrix} \;\middle|\; \begin{array}{l} \underline{s} \le s \le \overline{s}, \\ \underline{n}(s) \le n \le \overline{n}(s), \\ \underline{v_x} \le \dot{s}\,(1 - n\,C(s)) \le \overline{v_x}, \\ \underline{v_y} \le \dot{n} \le \overline{v_y}, \\ \underline{\dot{\psi}} \le C(s)\,\dot{s} \le \overline{\dot{\psi}}, \\ \underline{a_\psi} \le C'(s)\,\dot{s}^2 + C(s)\,u_t \le \overline{a_\psi}, \\ \begin{bmatrix} \underline{a_x} \\ \underline{a_y} \end{bmatrix} \le g(\tilde{x}_{di}, \tilde{u}_{di}) \le \begin{bmatrix} \overline{a_x} \\ \overline{a_y} \end{bmatrix}, \\ \|g(\tilde{x}_{di}, \tilde{u}_{di})\|^2 \le c. \end{array} \right\}. \tag{3.29}$$

The constraint set $\mathcal{C}$ is highly non-convex, primarily due to the curvature terms $C(s)$ and $C'(s)$ and their nonlinear interaction with the state and control inputs. To handle this non-convexity, we seek to derive a convex inner approximation of the feasible set $\mathcal{C}$. This objective is defined here and addressed in the following section.

Problem Definition: Finding an Inscribed Polytope

To handle the non-convexity, our approach is to approximate $\mathcal{C}$ with an inscribed polytope $\hat{C}$ that is convex. Formally, we seek to determine:

$$\hat{C} = \left\{ \begin{bmatrix} \tilde{x}_{di} \\ \tilde{u}_{di} \end{bmatrix} \;\middle|\; N \begin{bmatrix} \tilde{x}_{di} \\ \tilde{u}_{di} \end{bmatrix} \le b \right\} \subseteq \mathcal{C}, \tag{3.30}$$

where N and b represent a set of linear inequalities whose intersection forms the polytope. In the following section, we will demonstrate how to address this problem.

3.1.6 Convex Approximation via Quantifier Elimination

This section addresses the problem of determining an inscribed polytope, as defined in Problem 3.1.5. Our approach begins by simplifying the original constraint set through dimensionality reduction.

To achieve this, we decouple the state variables s, n, and $\dot{n}$, each of which is independently constrained within an interval:

$$\underline{s} \leq s \leq \overline{s}, \tag{3.31}$$
$$\underline{n}(s) \leq n \leq \overline{n}(s), \tag{3.32}$$
$$\underline{\dot{n}} \leq \dot{n} \leq \overline{\dot{n}}. \tag{3.33}$$

By ensuring that the remaining constraints hold for all values of s, n, and $\dot{n}$ within their respective intervals, we effectively reduce the problem to a lower-dimensional space involving only the remaining variables.

We define the reduced constraint set, $\tilde{C}$, over the variables $\dot{s}$ and the artificial control inputs $\tilde{u}_{di}$ as follows:

$$\tilde{C} = \left\{ \begin{bmatrix} \dot{s} \\ u_t \\ u_n \end{bmatrix} \;\middle|\; \begin{bmatrix} \tilde{x}_{di} \\ \tilde{u}_{di} \end{bmatrix} \in \mathcal{C}, \quad \forall \begin{bmatrix} s \\ n \\ \dot{n} \end{bmatrix} \in \begin{bmatrix} \underline{s}, \overline{s} \\ \underline{n}, \overline{n} \\ \underline{\dot{n}}, \overline{\dot{n}} \end{bmatrix} \right\}. \tag{3.34}$$

To make $\tilde{C}$ practically useful, we aim to express it as a set of linear inequalities. Since quantifiers cannot be directly modeled according to the DCP rules, we need to eliminate them.

The task of expressing $\tilde{C}$ as linear inequalities can be framed as a problem of **quantifier elimination**, a common technique in computer algebra. The goal of quantifier elimination is to transform a formula containing quantified variables into an equivalent form without quantifiers.

In our case, the constraint set can be expressed as the quantified formula:

$$\phi := \forall \begin{bmatrix} s \\ n \\ \dot{n} \end{bmatrix} \in \begin{bmatrix} \underline{s}, \overline{s} \\ \underline{n}, \overline{n} \\ \underline{\dot{n}}, \overline{\dot{n}} \end{bmatrix} : \quad \begin{aligned} & \underline{v_x} \leq \dot{s}(1 - nC(s)) \leq \overline{v_x} \quad \wedge \\ & \underline{\dot{\psi}} \leq C(s)\dot{s} \leq \overline{\dot{\psi}} \quad \wedge \\ & \underline{a_\psi} \leq C'(s)\dot{s}^2 + C(s)u_t \leq \overline{a_\psi} \quad \wedge \\ & \begin{bmatrix} \underline{a_x} \\ \underline{a_y} \end{bmatrix} \leq g(\tilde{x}_{di}, \tilde{u}_{di}) \leq \begin{bmatrix} \overline{a_x} \\ \overline{a_y} \end{bmatrix} \quad \wedge \\ & \|g(\tilde{x}_{di}, \tilde{u}_{di})\|^2 \leq c. \end{aligned} \tag{3.35}$$

Typically, quantifier elimination seeks a formula ϕ' such that:

$$\phi' \iff \phi$$

where ϕ' contains no quantifiers. However, in our case, we relax this requirement and allow ϕ' to be a sufficient condition for ϕ, meaning:

$$\phi' \implies \phi$$

This means that ϕ' describes a subset of $\tilde{C}$. This relaxation has two advantages:

It enables more efficient algorithms for quantifier elimination, as we do not need an exact equivalent formula, and it may be necessary, as $\tilde{C}$ is not guaranteed to be convex.

We propose two approaches to eliminate the quantifiers in ϕ and express $\tilde{C}$ as a set of linear inequalities. The first approach involves interval fitting for the state variables and control inputs, and we will present the results to facilitate replication. The second approach uses Cylindrical Algebraic Decomposition (CAD) to eliminate the quantifiers. Since CAD has multiple implementations and explaining this complex algorithm is beyond the scope of this work, we will only present the idea behind it by demonstrating the algorithm on an example. Finally, we will compare the results of both approaches and discuss their advantages and disadvantages.

Either approach will provide us with a polytope $\tilde{C}$, which we can use to define the solution to Problem:

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

Approach 1: Interval Fitting for State Variables and Control Inputs

The idea of the first approach 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 (3.35) 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.

$$c_{min} \leq f(x, y) \leq c_{max} \tag{3.37}$$

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:

$$f(x, y) = a(y)x + b(y) \tag{3.38}$$

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 $b(y)$:

$$a_{min} \leq a(y) \leq a_{max}, \quad b_{min} \leq b(y) \leq b_{max} \tag{3.39}$$

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

$$x \in [\underline{x}, \overline{x}] \implies \forall y \in Y : c_{min} \leq f(x, y) \leq c_{max} \tag{3.40}$$

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 (3.37) 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:

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

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

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 (3.37) 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:

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

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

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.

Approach 2: Cylindrical Algebraic Decomposition

The second approach is to use Cylindrical Algebraic Decomposition (CAD) to find the inscribed 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 [20, 21].

Example CAD

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 compliment 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.

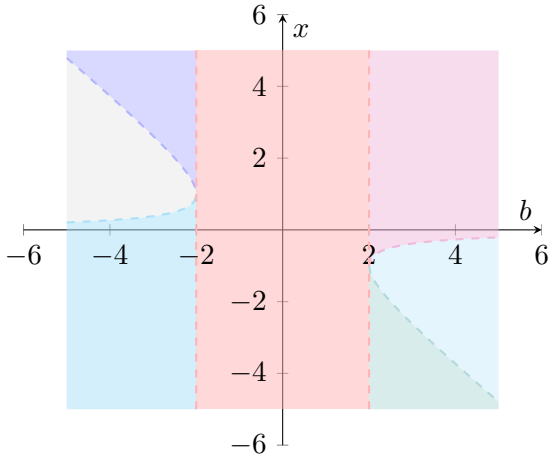


Figure 3.1: Illustrating the cells with shaded regions.

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 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.

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:

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

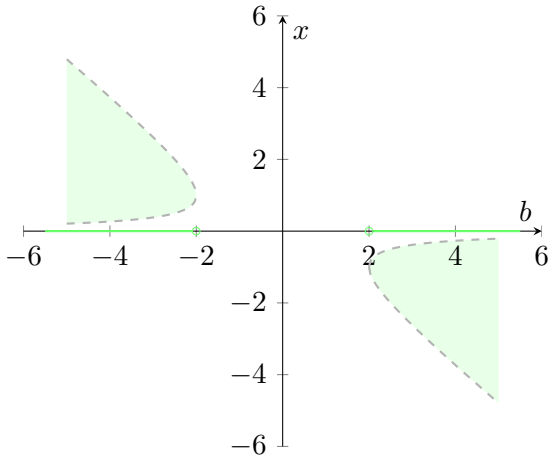


Figure 3.2: Illustrating the remaining cells.

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

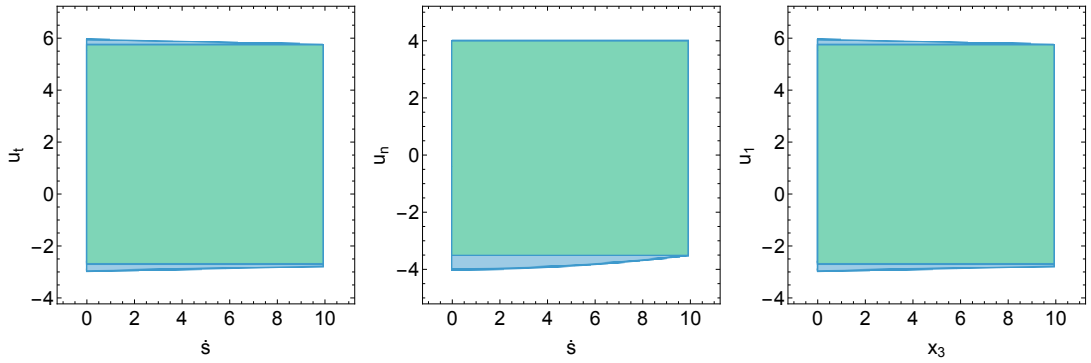
Comparison of Interval Fitting and CAD Approaches

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 Figure 3.3.

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.



(a) Region $\dot{s}$ vs u_t (b) Region $\dot{s}$ vs u_n (c) Region u_t vs u_n

Figure 3.3: In Green using Interval Fitting 3.1.6 and in Blue using CAD 3.1.6.

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.

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 [14]. A similar approach is demonstrated in [22], which served as an additional inspiration.

3.1.7 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.

$$v := v_x = \sqrt{(1 - nC(s))^2 \dot{s}^2 + \dot{n}^2} \tag{3.41}$$

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

$$\frac{\dot{n}}{\dot{s}} = (1 - nC(s)) \tan(\xi) = (1 - nC(s)) \tan(\psi - \theta) \tag{3.42}$$

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

$$\psi = \theta + \arctan\left(\frac{\dot{n}}{\dot{s}(1 - nC(s))}\right) \tag{3.43}$$

Using the state variables and g from (3.24), 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 $\cos \xi \neq 0$.

$$\dot{\psi} = \frac{a_{y,tn} - \tan(\xi) a_{x,tn}}{v(\tan(\xi)\sin(\xi) + \cos(\xi))} \tag{3.44}$$

$$a := a_x = \frac{a_{x,tn} + v\dot{\psi}\sin(\xi)}{\cos \xi} \tag{3.45}$$

Our Bicycle models (2.12) enables us to calculate the steering angle.

$$\delta = \arctan(l_{wb}\frac{\dot{\psi}}{v}) \tag{3.46}$$

With those equations, we can define a transformation.

$$T(\tilde{x}_{di}, \tilde{u}_{di}) = [p_x, p_y, \psi, \dot{\psi}, v, a, \delta] \tag{3.47}$$

3.1.8 Exact Discretization of the Double Integrator Model

Using the simplified state and control representation from Section 3.1.4, the system dynamics (3.16) are discretized using the matrix exponential method [23, 24]. The discrete-time system is formulated as:

$$\tilde{x}_{di,k+1} = A_d \tilde{x}_{di,k} + B_d \tilde{u}_{di,k} =: f_{d,di}(\tilde{x}_{di,k}, \tilde{u}_{di,k}, \Delta t), \tag{3.48}$$

where the exact discretization is given by:

$$A_d = e^{A\Delta t}, \quad B_d = \left(\int_0^{\Delta t} e^{A\tau} d\tau \right) B. \tag{3.49}$$

Using the closed-form solution for the matrix exponential [25], we obtain:

$$A_d = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad B_d = \begin{bmatrix} \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix}. \tag{3.50}$$

This formulation ensures that the discrete-time representation accurately follows the continuous-time system over a sampling interval Δt, preserving dynamic consistency while enabling efficient convex optimization. The resulting discrete-time model will be used in the subsequent trajectory planning formulation.

3.1.9 Final Model Representation

Our final model is represented by the following tuple.

$$M_{pm} = (\tilde{x}_{di}, \tilde{u}_{di}, f_{d,di}, \hat{C}, T) \tag{3.51}$$

3.2 Motion Planning using Bicycle Model

Instead of using the double integrator 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 (2.6) and (2.7).

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 (2.8)-(2.12).

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:

$$\dot{s} = \frac{v\cos\xi}{1 - nC(s)} \tag{3.52}$$

$$\dot{n} = v\sin\xi \tag{3.53}$$

$$\dot{\xi} = \dot{\psi} - C(s)\dot{s} \tag{3.54}$$

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 (3.55) and (3.56).

$$x_{kst} = \begin{bmatrix} s & n & \xi & v & \delta \end{bmatrix}^T \tag{3.55}$$

$$u_{kst} = \begin{bmatrix} a & v_\delta \end{bmatrix}^T \tag{3.56}$$

The dynamics of model are described by (3.57).

$$f_{kst}(x_{kst}, u_{kst}) = \begin{bmatrix} \frac{v\cos\xi}{1 - nC(s)} \\ v\sin\xi \\ \frac{1}{l_{wb}} v\tan\delta - C(s)\dot{s} \\ a \\ v_\delta \end{bmatrix}. \tag{3.57}$$

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 nonlinearity 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 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 linearization:

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

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. As discussed in Limitations and Outlook, segmenting the road model helps reduce conservatism in coupling constraints, a technique we previously applied to the double integrator model. In this context, another advantage is that we can model the curvature as a piece-wise linear function, which will be beneficial in subsequent steps.

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 $v_1 v_2$, 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 [26]. 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 v_1 and v_2 are bounded:

$$\underline{v_1} \leq v_1 \leq \overline{v_1}, \qquad \underline{v_2} \leq v_2 \leq \overline{v_2}$$

with constants $\underline{v_1}, \overline{v_1}, \underline{v_2}, \overline{v_2} \in \mathbb{R}$.

We introduce an auxiliary variable w to approximate the bilinear term $v_1 v_2$. Linear constraints are applied to bound w and ensure it accurately represents the bilinear term $v_1 v_2$. These constraints are derived from the bounds of the variables v_1, v_2, and the bilinear term $v_1 v_2$, as shown below:

$$\begin{aligned}
w &\geq \underline{v_1} v_2 + v_1 \underline{v_2} - \underline{v_1 v_2}, \\
w &\geq \overline{v_1} v_2 + v_1 \overline{v_2} - \overline{v_1 v_2}, \\
w &\leq \overline{v_1} v_2 + v_1 \underline{v_2} - \overline{v_1} \underline{v_2}, \\
w &\leq \underline{v_1} v_2 + v_1 \overline{v_2} - \underline{v_1} \overline{v_2}.
\end{aligned}$$

The idea behind these constraints is to create a convex lower bound and a concave upper bound for the bilinear term $v_1 v_2$. For example, the first bound is constructed as follows:

$$a = (v_1 - \underline{v_1}) \geq 0, b = (v_2 - \underline{v_2}) \geq 0$$

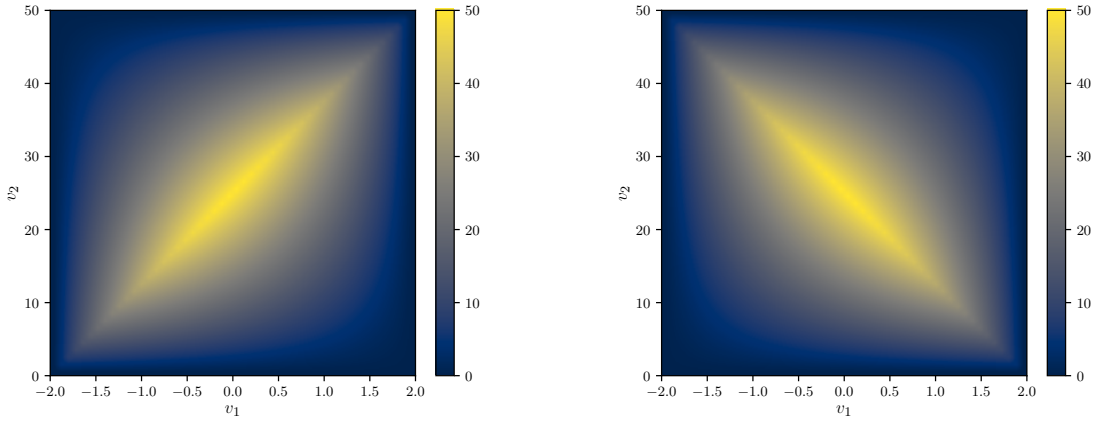
Since a and b are both non-negative, we can multiply them to get and obtain our first lower bound:

$$ab = v_1 v_2 - \underline{v_1} v_2 - v_1 \underline{v_2} + \underline{v_1 v_2} \geq 0 \iff v_1 v_2 \geq \underline{v_1} v_2 + v_1 \underline{v_2} - \underline{v_1 v_2}$$

To derive all possible lower and upper bounds, we can apply this pattern to $a \in \{v_1 - \underline{v_1}, \overline{v_1} - v_1\}$ and $b \in \{v_2 - \underline{v_2}, \overline{v_2} - v_2\}$. 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 $v_1 v_2$.

Figure 3.5a shows the difference to the upper bound, while Figure 3.5b shows the difference to the lower bound for the range $-2 \leq v_1 \leq 2$ and $0 \leq v_2 \leq 50$. It is evident that the bounds become tighter than v_1 and v_2 approach their respective limits.

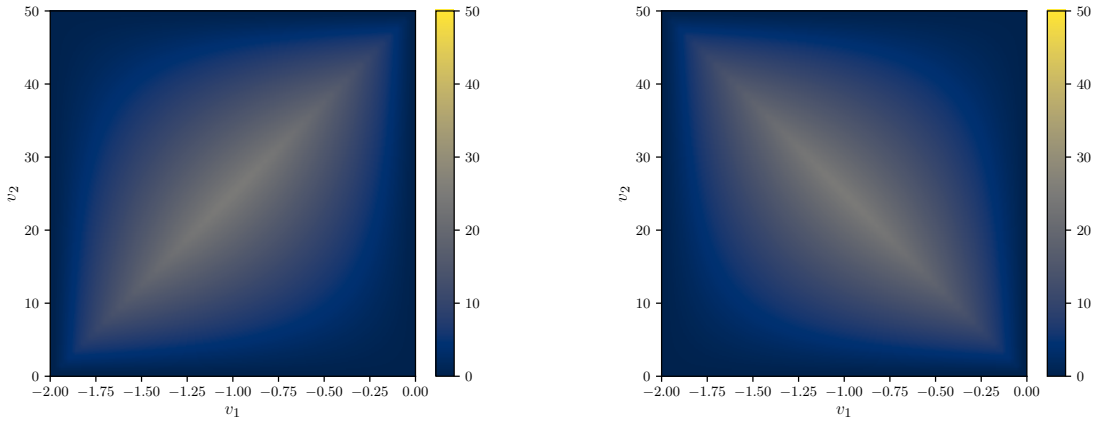
Figures 3.6a and 3.6b show the results when v_1 is more tightly bounded, specifically $-2 \leq v_1 \leq 0$. It is also clear that the maximum deviation is significantly reduced



(a) Difference to the upper bound

(b) Difference to the lower bound

Figure 3.5: McCormick relaxation bounds for the bilinear term $v_1 v_2$.



(a) Difference to the upper bound

(b) Difference to the lower bound

Figure 3.6: McCormick relaxation with tighter bounds on v_1.

compared to the previous scenario, demonstrating that tighter bounds lead to a more accurate relaxation.

To illustrate the application of these relaxations in practice, consider a path-planning scenario with $\underline{v} = 1$, $\overline{v} = 5$, 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 3.7 shows the planned velocity profile, which quickly reaches its upper limit.

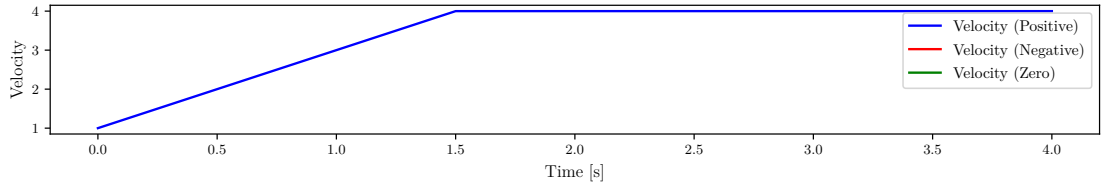


Figure 3.7: Planned velocity profile.

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.

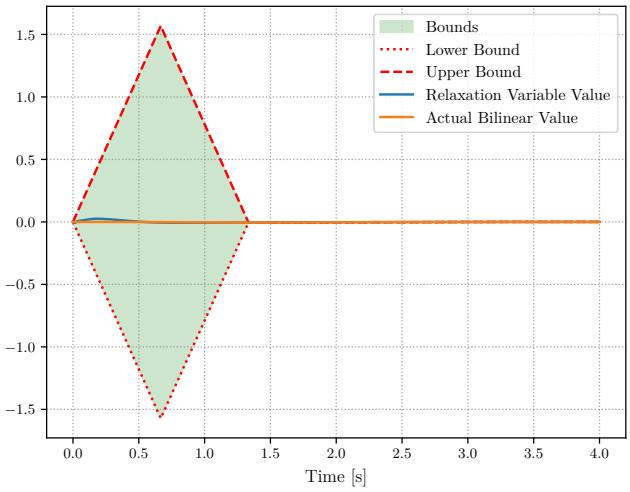


Figure 3.8: McCormick Relaxation on $v\xi$.

Figure 3.8 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.

For a more detailed discussion on McCormick relaxations and their applications in optimization, see [26, 27].

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:

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

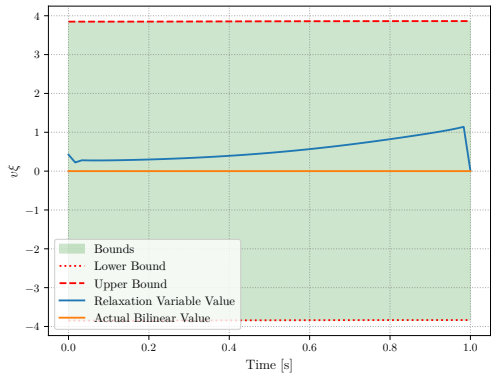
where Δt is the time step.

Of course, these bounds must always respect the physical velocity limits of the system, ensuring that $\underline{v} \leq v_i \leq \overline{v}$.

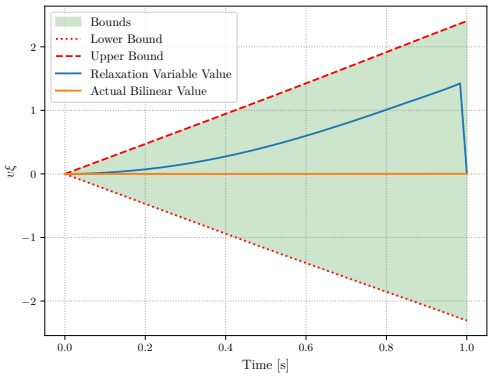
By iteratively applying these bounds for subsequent time steps, we can obtain increasingly precise constraints for the velocity. This leads to a tighter and more efficient McCormick relaxation for the bilinear terms.

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

Figure 3.9a illustrates the McCormick relaxation without tighter bounds, while Figure 3.9b 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 3.9: Comparison of McCormick relaxation with and without tighter bounds.

Since the accuracy of the relaxation depends on the tightness of these bounds, using broader constraints can result in significant approximation errors over time.

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:

$$\tilde{f}_{kst}(x_{kst}, u_{kst}) = \begin{bmatrix} v(\cos(\xi_0) + \sin(\xi_0)\xi_0) - \sin(\xi_0)w_{v,\xi} \\ v(\sin(\xi_0) - \cos(\xi_0)\xi_0) + \cos(\xi_0)w_{v,\xi} \\ \frac{v}{l_{wb}}(\tan(\delta_0) - \frac{\delta_0}{\cos^2(\delta_0)}) + \frac{w_{v,\delta}}{\cos^2(\delta_0)} - a_{i(s)}w_{s,\dot{s}} - b_{i(s)}\dot{s} \\ a \\ v_\delta \end{bmatrix} \tag{3.58}$$

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 3.10: McCormick relaxation constraints for the bilinear terms.

Coupling Constraints

The coupling constraints introduced by our discrete-time optimal trajectory problem 1.3.2 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:

$$s \in [\underline{s}, \overline{s}], \quad n \in [\underline{n}(s), \overline{n}(s)], \quad \xi \in [\underline{\xi}, \overline{\xi}], \quad v \in [\underline{v}, \overline{v}], \quad \delta \in [\underline{\delta}, \overline{\delta}] \tag{3.59}$$

$$v_\delta \in [\underline{v_\delta}, \overline{v_\delta}], \quad a \in \left[\underline{a}_{x,b},\, \overline{a}_{x,b} \min\left\{1, \frac{v_S}{v}\right\}\right] \tag{3.60}$$

where v_S is a parameter representing the switching velocity, which is used to encounter limiting engine power and breaking power, ensuring that acceleration remains feasible at different speeds, particularly at higher velocities where available power may be restricted.

The primary goal of the remaining section is to reformulate the friction circle constraint of our kinematic bicycle model (2.13) into a convex constraint, enabling efficient real-time optimization in control applications. The original constraint is given by:

$$\sqrt{a^2 + \left(v\dot{\psi}\right)^2} = \sqrt{a^2 + \left(\frac{v^2}{l_{wb}} \tan(\delta)\right)^2} \leq a_{max}. \tag{3.61}$$

Since this constraint contains nonlinear terms, particularly $\tan(\delta)$, it is not directly suitable for convex optimization. The following steps aim to approximate this constraint in a convex manner while maintaining physical validity.

Bounding the Steering Angle Nonlinearity

The term $\tan(\delta)$ introduces nonlinearity, making optimization difficult. To simplify, we use a conservative upper bound:

$$\tan(\delta) \leq \frac{\tan(\delta^*)}{\delta^*}\delta. \tag{3.62}$$

This transformation ensures:

- The constraint is easier to handle.
- The reformulation is conservative, meaning that feasible solutions under the new constraint remain valid for the original system.

Substituting this bound into the friction circle constraint gives a stricter approximation:

$$a^2 + \left(\frac{1}{l_{wb}} \frac{\tan(\delta^*)}{\delta^*}\right)^2 v^4 \delta^2 \leq a_{max}^2. \tag{3.63}$$

Introducing a Diamond-Shaped Overapproximation

The term $v^4\delta^2$ remains non-convex, so we introduce a convex upper bound by approximating it with a diamond-shaped constraint. The diamond shape is chosen because it provides a tight, convex overapproximation while being computationally efficient. We define a set of linear hyperplane constraints:

$$d_v v + d_\delta \delta \leq w, \tag{3.64}$$
$$d_v v - d_\delta \delta \leq w, \tag{3.65}$$
$$-d_v v + d_\delta \delta \leq w, \tag{3.66}$$
$$-d_v v - d_\delta \delta \leq w. \tag{3.67}$$

where the scaling factors $d_v = \frac{1}{v^*}$ and $d_\delta = \frac{1}{\delta^*}$ ensure the hyperplanes remain well-scaled with respect to the variables. Figure 3.12 illustrates how this diamond constraint tightly approximates the original non-convex term.

Deriving an Upper Bound for w

To ensure that the hyperplane approximation does not overly restrict the feasible set, we derive an upper bound $w(a)$. First, we express δ in terms of v by solving for δ in (3.63):

$$\delta = h(a)\frac{1}{v^2}, \quad \text{where } h(a) := \frac{l_{wb}\delta^*}{\tan(\delta^*)}\sqrt{a_{max}^2 - a^2}. \tag{3.68}$$

To find the tightest upper bound $w(a)$, we determine where the hyperplane is tangent to this curve. Taking the derivative of (3.68) with respect to v and equating it to the slope of the first hyperplane $-\frac{d_v}{d_\delta}$:

$$\frac{\partial \delta}{\partial v} = -2h(a)\frac{1}{v^3} \stackrel{!}{=} -\frac{d_v}{d_\delta} \quad \Rightarrow \quad v_{tight} = \sqrt[3]{2h(a)\frac{d_\delta}{d_v}}. \tag{3.69}$$

Substituting v_{tight} into the $\delta(v)$ equation (3.68) gives:

$$\delta_{tight} = h(a)\left(2h(a)\frac{d_\delta}{d_v}\right)^{-\frac{2}{3}}. \tag{3.70}$$

Finally, enforcing that $(v_{tight}, \delta_{tight})$ lies on the first hyperplane results in:

$$w(a) = d_v\sqrt[3]{2h(a)\frac{d_\delta}{d_v}} + d_\delta h(a)\left(2h(a)\frac{d_\delta}{d_v}\right)^{-\frac{2}{3}}. \tag{3.71}$$

Fitting a Convex Upper Bound for $w(a)$

Although $w(a)$ provides a tight bound, it may not be convex. To ensure convexity, we fit a function of the form:

$$\tilde{w}(a) = c_1 - \frac{1}{(|a| - c_2)^{2n}}, \tag{3.72}$$

where $c_1 = w(0)$ and $c_2 = a_{max} + c_1^{-\frac{1}{2n}}$. This choice ensures that $\tilde{w}(a)$ is a smooth, convex approximation of $w(a)$, making it well-suited for convex optimization solvers.

Based the parameters $\delta^* = 0.910$, $l_{wb} = 2.4$, and $a_{max} = 11.5$, we evaluated this heuristic approach on a velocity limit of $v^* = 14$. Figure 3.11 illustrates the function $w(a)$ and its approximation with $n = 2$ as a function of the acceleration a.

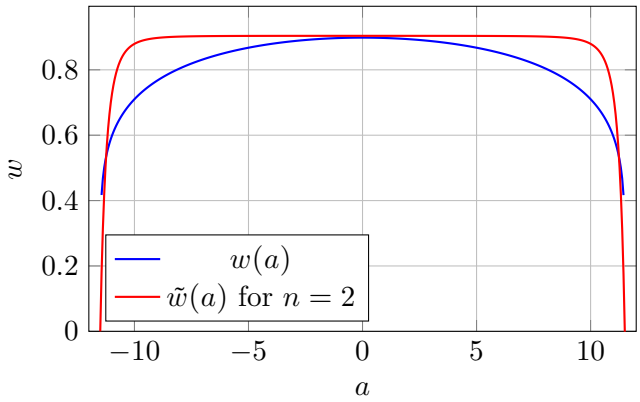


Figure 3.11: Plot of w versus a and its approximation.

The following Figure 3.12 visualizes the friction circle (3.61), its tighter constraint (3.62) 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:

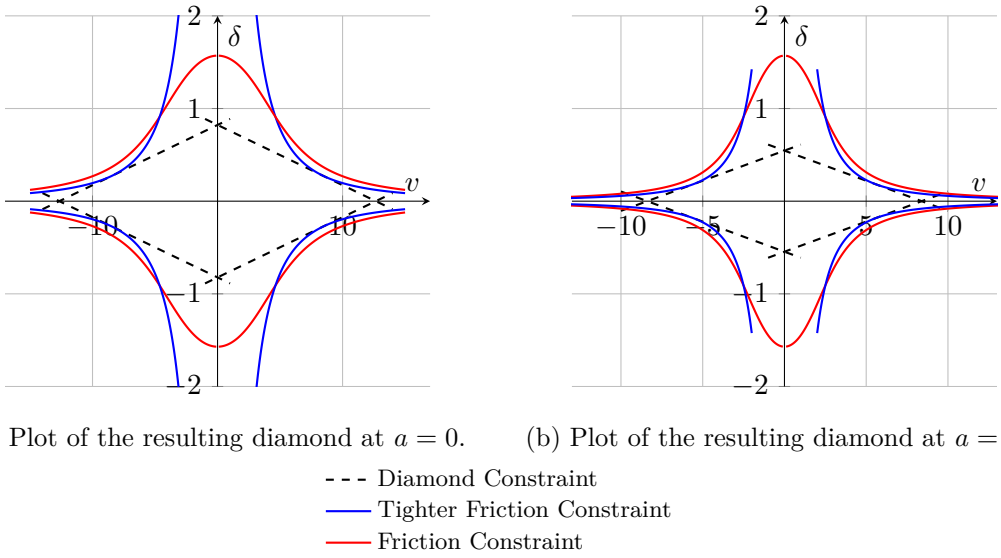
$$a^2 + (\frac{1}{l_{wb}} \frac{\tan(\delta^*)}{\delta^*})^2 w \leq a_{max}^2 \tag{3.73}$$

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

$$0 \leq w \leq \tilde{w}(a) \tag{3.74}$$

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

$$M_{kst} = (x_{kst}, u_{kst}, \tilde{f}_{kst}, \{w_{v,\xi}, w_{v,\delta}, w_{s,\dot{s}}, w\}, C) \tag{3.75}$$



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

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

Figure 3.12: Comparison of the resulting diamond constraints for different acceleration values.

where C consists of the coupling constraints (3.59), (3.60), the McCormick bounds (3.10), the introduced hyperplanes (3.64)-(3.67), and our approximated friction circle (3.73) with (3.74).

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} (3.51) and M_{kst} (3.75), 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

For our evaluation, we need to determine the road segment for each time point, configure our solver to solve the optimization problem, and provide the time points $\{t_i\}_{i=1,\dots,n}$.

4.1.1 Road Segments

For our double integrator 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 $\{t_i\}_{i=1,\dots,n}$. 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 $\{[s_{i-1}, s_i]\}_{i=1,\dots,m}$ 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:

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

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 $\{t_i\}_{i=0,\dots,n}$.

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:

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

The system dynamics differ based on the chosen model:

Double Integrator Model: Given the exact discretization of the double integrator model in (3.48), we define the system dynamics constraint as:

$$x(t_i) = f_{d,di}(x(t_{i-1}), u(t_{i-1}), t_i - t_{i-1}) \tag{4.3}$$

Bicycle Model: Since the dynamics of the kinematic bicycle model include auxiliary variables, we apply forward Euler discretization to the continuous-time dynamics:

$$x(t_i) = x(t_{i-1}) + \tilde{f}_{kst}(x(t_{i-1}), u(t_{i-1})) \cdot (t_i - t_{i-1}) \tag{4.4}$$

where $\tilde{f}_{kst}$ is defined in (3.58).

These system dynamics, combined with the coupling constraints and additional constraints, define the full set of variables and constraints used by the 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.

While our planner primarily relies on hard constraints to ensure feasibility and adherence to system dynamics, real-world scenarios often introduce uncertainties, disturbances, or conflicting objectives that make strict constraint satisfaction impractical. To address this, we incorporate soft constraints, which allow for controlled constraint relaxation in exchange for a penalty. By integrating soft constraints into the optimization framework, we can balance feasibility with optimality, enabling more flexible and robust motion planning.

4.1.3 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.

4.1.4 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.

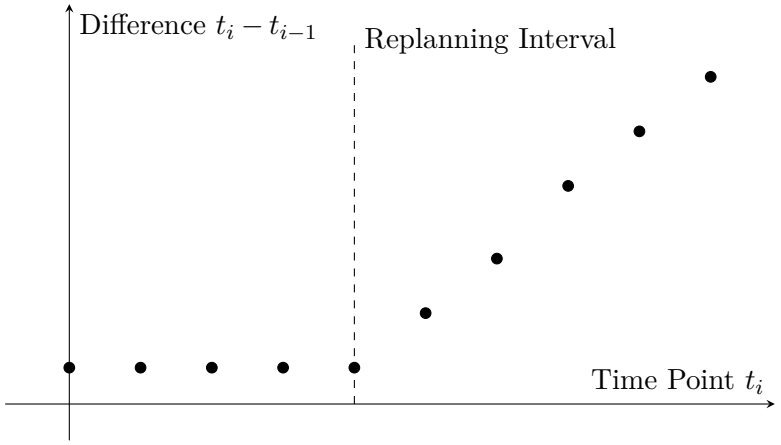


Figure 4.1: Time points and their differences to previous time points

Figure 4.1 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 $\Delta^2 t_{replan}$.

Next, we introduce our benchmarking framework, consisting of configurations, such as objectives, road.

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:

$$J_{control} = \sum_{i=0}^{n-1} \|d_1(t_i)\|^2 \tag{4.5}$$

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:

$$J_{tracking} = \sum_{i=0}^{n} d_2(t_i)^2 \tag{4.6}$$

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:

$$J_{terminal} = \|x(t_n) - x_{final}\|^2 \tag{4.7}$$

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

$$J = \alpha J_{control} + \beta J_{tracking} + \gamma J_{terminal} \tag{4.8}$$

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.

Since some of our objectives, such as $J_{tracking}$, involve nonlinear or nonconvex formulations, directly incorporating them into the optimization problem can be challenging. To address this, we introduce auxiliary variables that allow us to reformulate certain objectives into convex, computationally efficient expressions. These auxiliary variables help model constraints and cost functions in a way that preserves convexity while maintaining the intended optimization behavior.

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:

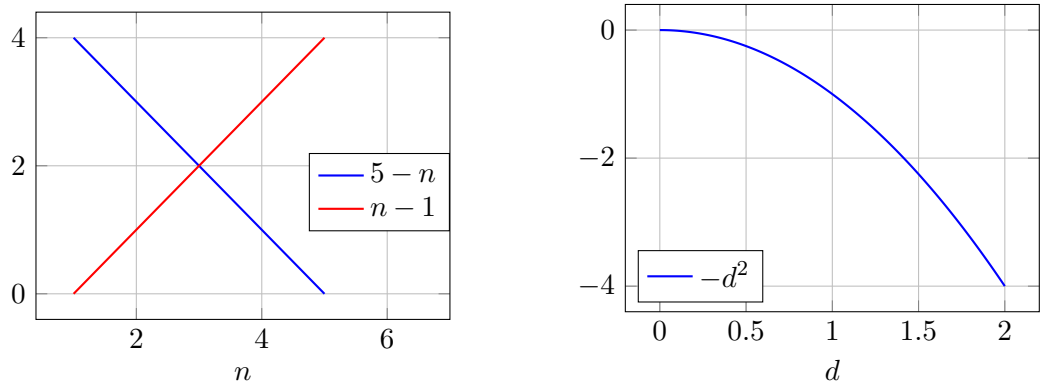
$$0 \leq d \leq \min \{\overline{n}(s) - n, n - \underline{n}(s)\}$$

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 4.2. Let's assume $\overline{n}(s) = 5$ and $\underline{n}(s) = 1$.



(a) Distance to Road Boundaries

(b) Objective Function

Figure 4.2: 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.

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:

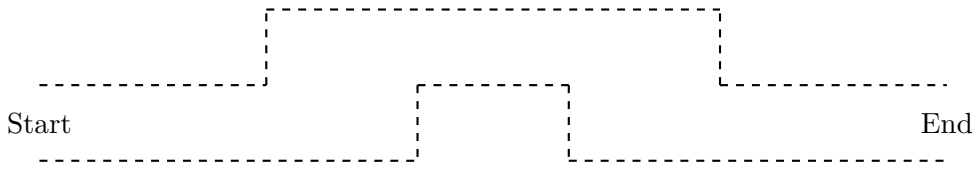


Figure 4.3: Elchtest scenario visualization

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]
	4	30.0	0.000	[-2.0,2.0]	[-2.0,2.0]
Slalom	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]
	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]
Feasible Curve	1	20.0	0.000	[-2.0,2.0]	[-2.0,2.0]
	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]
Infeasible Curve	1	20.0	0.000	[-2.0,2.0]	[-2.0,2.0]
	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]

Table 4.1 provides an overview of the road segments used in our evaluation scenarios. Each segment is characterized by its length, curvature, and lane width at the start and end points. This detailed breakdown helps in understanding the specific challenges posed by each scenario and how the trajectory planner adapts to different road conditions.

4.2.3 Simulation Setup

For the vehicle simulation, we employ a more sophisticated model from [18] 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 $|v| \geq 0.1$:

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

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

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

We consider a vehicle, with the identifier '1' from [18] of length $l = 4.298\,\text{m}$ and width $w = 1.674\,\text{m}$, with total mass $m = 1.225 \times 10^3\,\text{kg}$ and moment of inertia $I_z = 1.538 \times 10^3\,\text{kg}\,\text{m}^2$. The center of gravity is located $l_f = 0.883\,\text{m}$ from the front axle and $l_r = 1.508\,\text{m}$ from the rear axle, at a height $h_{cg} = 0.557\,\text{m}$. The front and rear cornering stiffness coefficients are both $C_{S,f} = C_{S,r} = 20.89\,[1/\text{rad}]$, and the friction coefficient is $\mu = 1.048$. The switching velocity for the dynamics is set to $v_S = 4.755\,\text{m/s}$.

4.2.4 Results

Throughout our simulations, we defined specific ranges for the control inputs to ensure realistic vehicle behavior. The longitudinal acceleration, a_x, was constrained within $[-6, 3]$ m/s², while the steering rate, v_δ, was limited to $[-0.5, 0.5]$ rad/s. Additionally, the steering angle, δ, was bounded within $[-0.698, 0.698]$ radians.

$$a_x \in [-6, 3]\ \text{m/s}^2, \quad v_\delta \in [-0.5, 0.5]\ \text{rad/s}, \quad \delta \in [-0.698, 0.698]\ \text{rad}$$

To evaluate performance under different conditions, we simulated all scenarios at three distinct speeds:

$$v_{low} = 5\text{m/s}, v_{mid} = 10\text{m/s}, \text{and } v_{high} = 20\text{m/s}.$$

We also allowed the vehicle to decelerate down to 70% of its initial speed.

For time discretization, we implemented two configurations, represented as

$$t_{\text{conf}} = (T, R, \Delta t, \Delta^2 t_{\text{replan}})$$

where T is the time horizon, R is the replanning interval, the initial constant time interval Δt for the first few time points, and the increasing time interval $\Delta^2 t_{\text{replan}}$ for the remaining time points as illustrated in 4.1. The first configuration, $t_{\text{conf}}^{(1)}$, was set to a smaller time horizon with a finer Δt, while the second configuration, $t_{\text{conf}}^{(2)}$, used a larger time horizon with a coarser Δt as well as a smaller slope for the time steps after the replanning interval, providing two distinct approaches for evaluating planning and control strategies.

$$t_{\text{conf}}^{(1)} = (3s, 0.1s, 10ms, 40ms)$$
$$t_{\text{conf}}^{(2)} = (5s, 0.1s, 20ms, 20ms)$$

We used four objectives to evaluate performance: the control effort cost $J_{control}$, the trajectory tracking cost $J_{tracking}$, the terminal cost $J_{terminal}$, and the combined cost function J from (4.8), with weighting factors $\alpha = 1$, $\beta = 10^3$, and $\gamma = 10^4$. Those weights were chosen to equally balance the objectives.

$$J_{control}, J_{tracking}, J_{terminal}, J$$

All simulations were conducted on a MacBook Air equipped with an Apple M1 processor and 16 GB of unified memory. The operating system used was macOS 15.3 (24D60). The simulations were executed using Python 3.11.3, compiled with Clang 13.0.0.

Solver Times

This section evaluates solver performance across different models and configurations. We assessed efficiency by running simulations with varying velocity, scenarios, and objective functions, totaling 96 simulations per model-configuration pair.

Table 4.2 summarizes the average solver time and its deviation for each model and configuration.

Table 4.2: Solver Performance for Different Models and Configurations

Model	**Configuration**	**Avg Time (ms)**	**Time Deviation (ms)**
Double Integrator	$t_{\text{conf}}^{(1)}$	3.9	1.0
Double Integrator	$t_{\text{conf}}^{(2)}$	3.8	1.3
Bicycle	$t_{\text{conf}}^{(1)}$	9.5	2.1
Bicycle	$t_{\text{conf}}^{(2)}$	9.4	2.9

The double integrator model outperforms the bicycle model, achieving solver times of 3.9ms and 3.8ms across both configurations. In contrast, the bicycle model requires 9.5ms and 9.4ms, making it over twice as slow. Solver time deviations also differ significantly.

This indicates that the double integrator model provides not only faster solutions but also more stable performance.

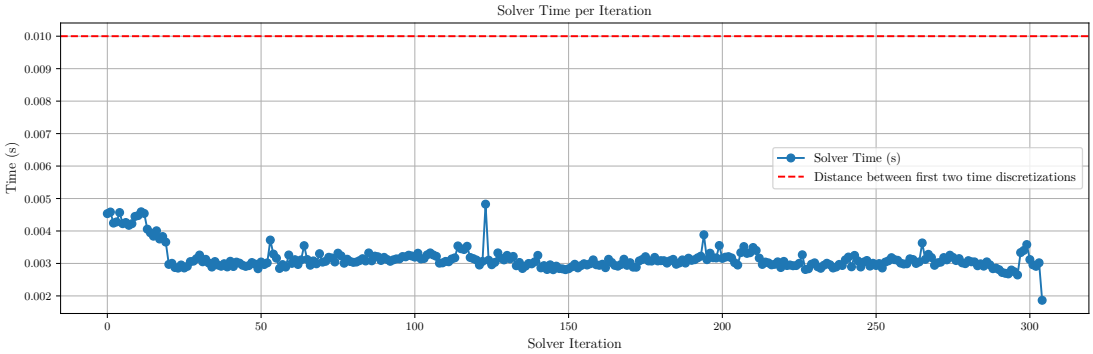


Figure 4.4: Solver metrics for Slalom scenario using double integrator model

Figures 4.4 and 4.5 illustrate solver performance in the Slalom scenario.

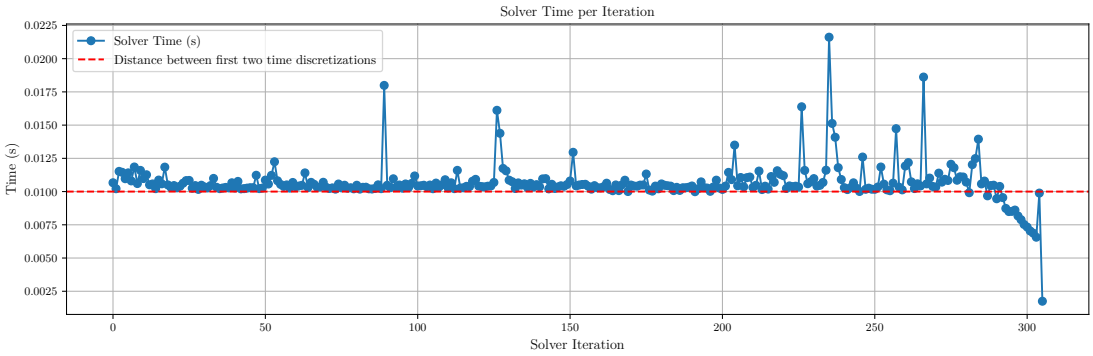


Figure 4.5: Solver metrics for Slalom scenario using kinematic bicycle model

- The double integrator model maintains relatively stable solver times across iterations.
- The bicycle model exhibits more variation, aligning with the larger solver time deviations observed in Table 4.2.

These findings confirm that the double integrator model provides a more consistent and efficient solution.

Completion Rates

Figure 4.6 presents the number of failed scenarios per model. As expected, both models fail every test in the Infeasible Curve scenario, which is intentionally designed to be unsolvable. However, when the curve radius increases slightly (making the scenario marginally feasible), both models successfully complete it at the lowest velocity v_{low}.

The results indicate that higher speeds lead to higher failure rates, as expected. However, an anomaly occurs in the Straight scenario, where the bicycle model has a higher failure rate at lower velocities. This issue arises when using the J_{terminal} objective function, which prioritizes velocity maximization. This suggests a numerical instability that may be resolved using soft constraints.

The bicycle model completes more runs in the slalom scenario compared to the double integrator model. This is because the double integrator model considers the worst-case scenario and does not find a solution if it is infeasible to drive on the inner side of the curve, even though it might be feasible to drive on the outer line. For example driving at the highest speed v_{max}, the resulting polytope of the point mass model is empty. Limiting the vehicle options to the outer side and reduce the upper speed limit to 14.5m/s results in a non-empty set. In fact, we can observe that the bicycle model completes the slalom always at the outer lines (see 4.8), while driving at around 14.5m/s (see 4.9).

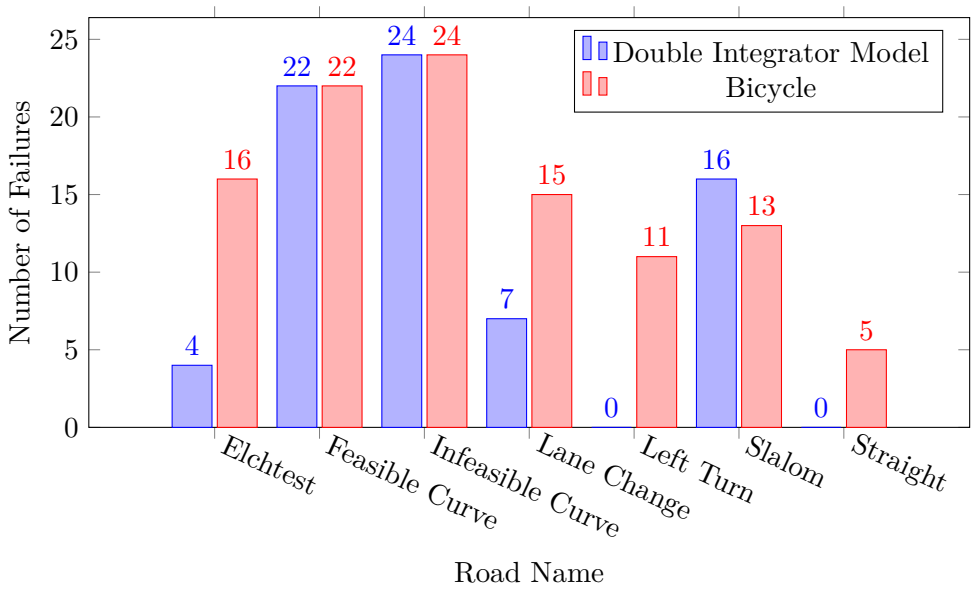


Figure 4.6: Histogram of Failed Scenarios per Model

Overall, the kinematic bicycle model exhibits a higher failure rate, particularly in the Elchtest and Lane Change scenarios at high speeds, where sharp turns are necessary. This is primarily due to the limitations imposed by the friction circle approximation, which the model struggles to handle effectively. In contrast, the double integrator model performs better in these scenarios as it does not depend on the friction circle approximation. The double integrator model incorporates this constraint statically, whereas the dynamic approach of the bicycle model allows it to choose between higher speeds with reduced steering capability or lower speeds with increased steering capability. Further investigation into this behavior is warranted.

Friction circle

We can calculate the constraints on the steering angle for v_{high}, resulting in a steering angle range of $\delta \in \emptyset$, while the exact range for the model is $\delta \in [-0.07, 0.07]$. For v_{mid}, the exact range is $\delta \in [-0.26, 0.26]$, whereas the approximated range is $\delta \in [-0.21, 0.21]$. Without considering the friction circle, the model completes all simulations but violates the friction circle constraints. This makes high-speed maneuvers infeasible for the bicycle model, as the friction circle approximation is too restrictive to accommodate the required steering angles.

Using $J_{terminal}$, the model tries to maximize its velocity, exacerbating the problem since it does not slow down to achieve a higher steering angle range. This issue can be

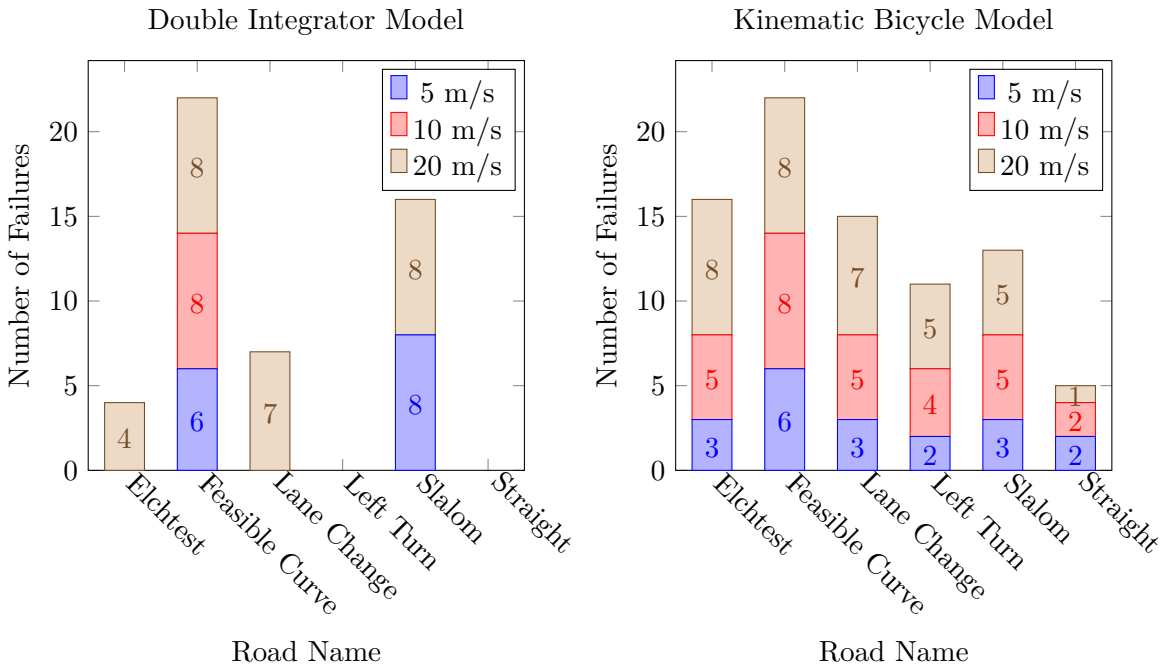


Figure 4.7: Stacked Histogram of Failed Road Names per Model with Velocity

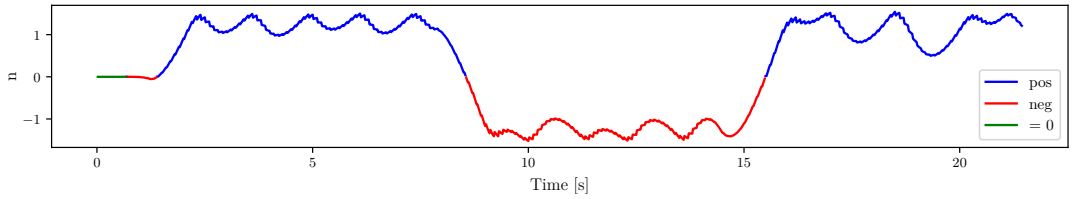


Figure 4.8: Later offset for slalom using bicycle model

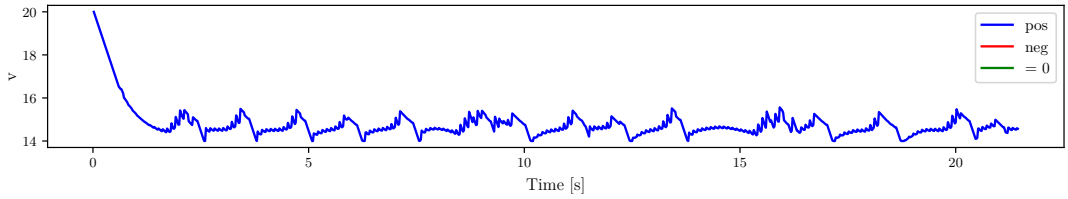


Figure 4.9: Velocity for slalom using bicycle model

mitigated by introducing soft constraints, but the model will still attempt to keep the steering angle as low as possible. In the left turn scenario, this results in a decreasing

lateral offset. Combined with the small angle approximation inaccuracies and ignoring forces during planning, the vehicle ends up hitting the right boundary of the road.

To address this, we introduce a default constraint with an additional auxiliary variable v:

$$\min\{\overline{n}(s(t_i)) - n(t_i), n(t_i) - \underline{n}(s(t_i))\} \geq c - v$$

where c is a constant modeling the desired distance to the road boundary, and $v \geq 0$ converts the constraint into a soft constraint. This way, the model can still try to minimize the cost function but must pay a penalty for getting too close to the road boundary. Figure 4.10 shows this behavior, where we added the soft constraint to the

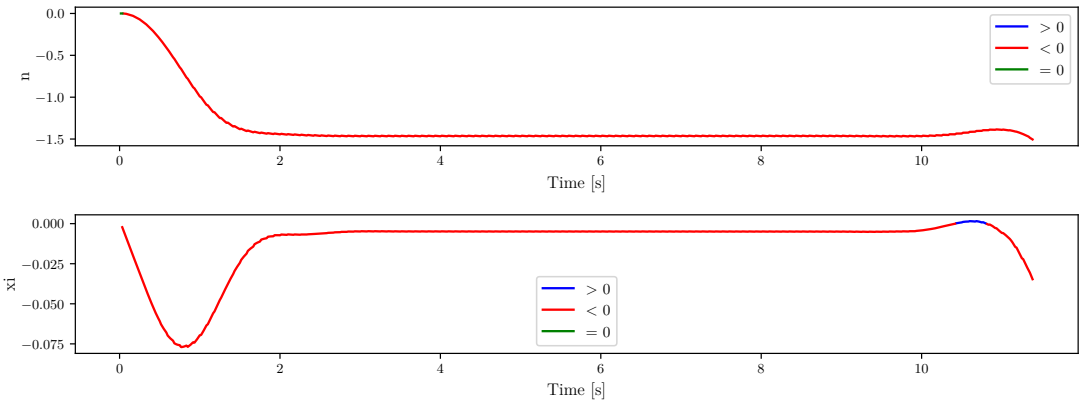


Figure 4.10: Predictive model states for lane change using bicycle model

left turn scenario, with $c = 0.5$.

5 Conclusion and Future Work

5.1 Conclusion

In this thesis, we investigated predictive planning and control strategies using two distinct vehicle models (a double integrator model and a kinematic bicycle model) under a variety of road scenarios and velocity conditions. Our principal objectives were to analyze solver performance, quantify completion rates, and examine how friction-circle constraints affect high-speed maneuvers.

Solver Times The double integrator model consistently outperformed the bicycle model in terms of solver time, requiring on average around 3.8–3.9,ms compared to 9.4–9.5,ms for the bicycle model under both time discretization configurations. These results also demonstrated that the double integrator model exhibited more stable performance, indicated by smaller time deviations.

Completion Rates With respect to scenario feasibility, both models failed all runs of the intentionally infeasible curve scenario. When the curve was enlarged to a marginally feasible radius at lower speeds, both models succeeded. As speeds increased, failure rates rose across the board, underscoring the critical impact of higher velocities on path-following constraints. Notably, the bicycle model faced more failures in sharp-turn scenarios (Elchtest, Lane Change), especially at high speeds, mainly due to restrictive friction-circle approximations. Conversely, the double integrator model, while simpler, proved more robust in these scenarios because it does not dynamically couple speed and steering to friction limits.

Friction-Circle Constraints Our investigations into the friction circle revealed that high-speed maneuvers became infeasible for the bicycle model under strict friction approximations: once the model selects a higher velocity, allowable steering angles shrink, making sharp maneuvers impossible. The terminal cost function further exacerbated this by incentivizing high speed at the expense of feasible steering control. By introducing soft constraints on road-boundary proximity, some numerical instabilities were alleviated, but the fundamental trade-off between maintaining high speed and ensuring adequate steering authority remained a challenge.

Overall, the double integrator model achieved faster and more stable solver performance, but at the cost of less physical realism regarding friction limits. The kinematic bicycle model provided a more accurate vehicle representation, but suffered from higher solver times and increased failure rates in tightly constrained, high-speed scenarios.

5.2 Future Work

While the presented results underline the strengths and weaknesses of both models, several avenues for future work arise:

- **Improved Friction Modeling:** The friction-circle approximation can be refined by incorporating velocity-dependent tire models or more sophisticated dynamics (e.g., Pacejka models). These enhancements would reduce infeasibilities at high speeds, but may require advanced solvers or tailored approximations to maintain computational efficiency.
- **Adaptive Time Discretization:** Although two time-discretization configurations were explored, a more adaptive approach could dynamically adjust the horizon or time-step sizes during runtime based on scenario complexity or solver convergence metrics. This may further optimize computational resources, especially in high-speed maneuvers.
- **Soft Constraints and Penalty Tuning:** Our preliminary introduction of soft constraints highlights the potential for balancing feasibility against other objectives. Systematic tuning of penalty weights (e.g., for road-boundary offsets) and investigating robust methods to handle constraint violations could improve success rates for the bicycle model.
- **Multi-Stage Approaches:** A hierarchical or multi-stage planning approach (e.g., generating coarse global paths first, then refining with a local high-fidelity model) could blend the computational simplicity of the double integrator model with the accuracy of the bicycle model. This may reduce the need for global friction-circle approximations without sacrificing real-world realism.
- **Hardware Deployment and Real-Time Performance:** Deploying the proposed methods on embedded hardware or real test vehicles remains an important step. Evaluating trade-offs between model fidelity and real-time feasibility could guide the choice of model and solver in production systems.
- **Extended Cost Functions:** Future extensions might include advanced cost functions considering comfort, energy efficiency, and safety margins. Additionally, refining velocity maximization objectives to account for friction and steering

constraints explicitly may prevent infeasibilities observed in certain high-speed or narrow-lane scenarios.

By addressing these areas, subsequent research can build upon the findings of this thesis to develop more robust, efficient, and realistic motion planners. The interplay between solver performance, model fidelity, and scenario constraints remains an active and critical frontier in predictive control for autonomous driving.

List of Figures

List of Tables

Bibliography

[1] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. "A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles." In: *IEEE Transactions on Intelligent Vehicles* 1.1 (Mar. 2016). Conference Name: IEEE Transactions on Intelligent Vehicles, pp. 33–55. ISSN: 2379-8904.

[2] S. Thrun et al. "Stanley: The robot that won the DARPA Grand Challenge." In: *J. Field Robotics* 23 (Jan. 2006), pp. 661–692.

[3] M. Montemerlo et al. "Junior: The Stanford Entry in the Urban Challenge." In: *Journal of Field Robotics* 25 (Sept. 2008), pp. 569–597. ISSN: 978-3-642-03990-4.

[4] N. D. Van, M. Sualeh, D. Kim, and G.-W. Kim. "A Hierarchical Control System for Autonomous Driving towards Urban Challenges." en. In: *Applied Sciences* 10.10 (Jan. 2020). Number: 10 Publisher: Multidisciplinary Digital Publishing Institute, p. 3543. ISSN: 2076-3417.

[5] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser. "A Review of Motion Planning for Highway Autonomous Driving." In: *IEEE Transactions on Intelligent Transportation Systems* 21.5 (May 2020). Conference Name: IEEE Transactions on Intelligent Transportation Systems, pp. 1826–1848. ISSN: 1558-0016.

[6] D. Gonzalez Bautista, J. Pérez Rastelli, V. Milanes, and F. Nashashibi. "A Review of Motion Planning Techniques for Automated Vehicles." In: *IEEE Transactions on Intelligent Transportation Systems* 17 (Nov. 2015), pp. 1–11.

[7] A. Orthey, C. Chamzas, and L. E. Kavraki. "Sampling-Based Motion Planning: A Comparative Review." en. In: *Annual Review of Control, Robotics, and Autonomous Systems* 7.Volume 7, 2024 (July 2024). Publisher: Annual Reviews, pp. 285–310. ISSN: 2573-5144.

[8] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel. *Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization.* June 2013.

[9] R. Tedrake. "Trajectory Optimization." In: *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation.* MIT CSAIL, 2023. Chap. 10.

[10] P. Falcone, F. Borrelli, J. Asgari, E. Tseng, and D. Hrovat. "Predictive Active Steering Control for Autonomous Vehicle Systems." In: *Control Systems Technology, IEEE Transactions on* 15 (June 2007), pp. 566–580.

[11] A. Gray, Y. Gao, J. K. Hedrick, and F. Borrelli. "Robust Predictive Control for semi-autonomous vehicles with an uncertain driver model." In: *2013 IEEE Intelligent Vehicles Symposium (IV)*. ISSN: 1931-0587. June 2013, pp. 208–213.

[12] T. Xia and H. Chen. "A Survey of Autonomous Vehicle Behaviors: Trajectory Planning Algorithms, Sensed Collision Risks, and User Expectations." en. In: *Sensors* 24.15 (Jan. 2024). Number: 15 Publisher: Multidisciplinary Digital Publishing Institute, p. 4808. ISSN: 1424-8220.

[13] K. Esterle, T. Kessler, and A. Knoll. "Optimal Behavior Planning for Autonomous Driving: A Generic Mixed-Integer Formulation." In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. ISSN: 2642-7214. Oct. 2020, pp. 1914–1921.

[14] 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.

[15] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli. "Kinematic and dynamic vehicle models for autonomous driving control design." In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. ISSN: 1931-0587. June 2015, pp. 1094–1099.

[16] P. Polack, F. Altché, B. D'Andrea-Novel, and A. de La Fortelle. "Guaranteeing Consistency in a Motion Planning and Control Architecture Using a Kinematic Bicycle Model." In: *2018 Annual American Control Conference (ACC)*. ISSN: 2378-5861. June 2018, pp. 3981–3987.

[17] J. H. Reif. "Complexity of the mover's problem and generalizations." In: *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. ISSN: 0272-5428. Oct. 1979, pp. 421–427.

[18] *Dateien · master · tum-cps / commonroad-vehicle-models · GitLab*. de. Jan. 2021.

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

[20] 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.

[21] 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.

[22] 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.

[23] T. Kailath. "Linear Systems." In: Jan. 1980.

[24] K. Ogata. *Modern control engineering*. Jan. 2010. ISBN: 978-0-13-615673-4.

[25] *Matrix Analysis for Scientists and Engineers | SIAM Publications Library*. en.

[26] G. P. McCormick. "Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems." en. In: *Mathematical Programming* 10.1 (Dec. 1976), pp. 147–175. ISSN: 1436-4646.

[27] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. "Branching and bounds tightening techniques for non-convex MINLP." In: *Optimization Methods and Software* 24 (Oct. 2009), pp. 597–634.