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: | 01.01.2019 |

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, 01.01.2019 Andreas Belavic

Acknowledgments

A thesis may include an acknowledgments section where the author may want to thank certain people, groups, institutions, etc. who provided support.

# Abstract

This is the abstract. It is a short summary of your work, consisting of roughly one to three paragraphs. It should give the main ideas of your paper, i.e., the posed problem, a motivation for solving it, your solution method, and your results results results. Keep it understandable for a general audience. Do not include references.

# Contents

# 1 Introduction

## 1.1 Introduction

The problem of autonomous driving can be divided into four main components, each representing a crucial aspect of the system.

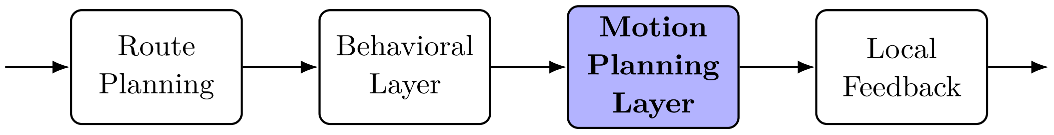


Figure 1.1: Overview of Autonomous Driving Problem Decomposition

The process begins with the user providing a travel destination, which serves as the input to the first component, Route Planning. In this phase, the system generates a sequence of waypoints through a predefined road network.

Next, the Behavioral Layer refines the waypoints by considering environmental factors such as other vehicles, obstacles, and road signs. This layer defines the driving behavior at any given time, ensuring the vehicle adapts to dynamic traffic conditions.

With a behavior strategy in place, the Motion Planning Layer generates a trajectory that adheres to strict physical and safety constraints, ensuring feasibility and compliance with rules of the road.

Finally, the Local Feedback component executes the plan by generating precise control commands—steering, throttle, and brake inputs—based on real-time vehicle and environmental feedback.

Finding an exact solution to the motion planning problem is computationally intractable in most cases. Therefore, numerical approaches are employed. These methods fall into three main categories:

1. Graph-Based Algorithms: These algorithms discretize the vehicle's possible states and connect valid state transitions with edges. A graph search algorithm can then be used to find an optimal trajectory.

2. Incremental Tree Approaches: This category involves generating branches of feasible trajectories by randomly applying control commands and simulating the resulting

states. The tree expands incrementally, exploring potential paths until a suitable trajectory is found.

3. Optimization-Based Methods: These methods formulate the problem as an optimization task over a function space. Optimization techniques aim to minimize an objective function (e.g., minimizing travel time or maximizing safety) while respecting constraints. This is the approach we focus on.

Our objective is to design a motion planner that consistently provides near real-time solutions. We achieve this by leveraging modern, reliable solvers that can efficiently handle the optimization problem.

## 1.2 Problem Formulation

The problem of motion planning using an optimization-based approach is defined as follows:

The objective is to find a function $\pi(t) : [0, T] \to \mathcal{X}$, where $T$ represents the planning horizon and $\mathcal{X}$ is the configuration space, which defines the set of feasible configurations for the vehicle.

The vehicle starts in an initial configuration $x_{\text{initial}} \in \mathcal{X}$, and the trajectory should end in a set of goal configurations $X_{\text{goal}} \subset \mathcal{X}$.

Additionally, a constraint is defined over the derivatives of $\pi$ with respect to $t \in [0, T]$, denoted as $D(\pi(t), \pi'(t), \pi''(t), \dots)$.

To formulate an optimization problem, we define an objective function and a feasible set.

Let $\Pi[\mathcal{X}, T]$ represent the set of all possible functions mapping $[0, T]$ to $\mathcal{X}$. Further, let $J(\pi) : \Pi[\mathcal{X}, T] \to \mathbb{R}$ be the objective function.

### Problem Definition: Optimal Trajectory Planning

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

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

## 1.3 Related Work

## 1.4 Thesis Structure

# 2 Preliminaries

## 2.1 Convex Discrete-Time Optimization

### 2.1.1 Complexity

Finding an exact solution in a dynamic environment is highly challenging. The problem is inherently non-convex, and solvers cannot directly operate over a function space.

### 2.1.2 Numerical Methods

To address this problem numerically, we first define the constraints by modeling the vehicle and its environment. We then reformulate the problem, discretize it, and approximate it. Our goal is to obtain a solution that is both computationally efficient and reliable.

To achieve this, we employ a convex solver, necessitating adherence to disciplined convex programming (DCP) rules.

### 2.1.3 Variational Method

#### Direct Methods

To project the function space into a finite-dimensional vector space, we define a set of basis functions that span a subspace of the original space. A function in this subspace is then represented as a linear combination of the basis functions:

$$\tilde{\pi}(t) = \sum_{i=1}^{N} \pi_i \phi_i(t) \tag{2.1}$$

While this approach restricts the search space, potentially deviating from the true optimal solution, it allows for numerical tractability. One of the most widely used techniques for trajectory approximation is numerical integration with collocation. In this approach, the trajectory is required to satisfy the constraints at a discrete set of time points $\{t_i\}_{i=1}^{m}$. Numerical integration techniques are then employed to approximate the trajectory between these points.

### Discrete-Time Problem Formulation

Using direct methods, we now reformulate the optimization problem over a finite-dimensional vector space.

Let $\mathcal{X}$ be the set of valid vehicle states and $\mathcal{U}$ the set of all feasible control inputs.

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

### Problem Definition: Discrete-Time Optimal Trajectory Planning

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

$$
\begin{aligned}
u^* = \underset{u \in \mathcal{U}^{T-1}}{\arg\min} \sum_{i=1}^{T-1} J(x_{i+1}, u_i) & & \quad (2.2)\\
\text{s.t.} \quad x_1 = x_{\text{initial}} & & \quad (2.3)\\
x_T \in X_{\text{goal}} \subseteq \mathcal{X} & & \quad (2.4)\\
(x_i, u_i) \in \mathcal{C} \subseteq \mathcal{X} \times \mathcal{U} & \quad \forall i \in \{1, \dots, m-1\} & \quad (2.5)\\
x_{i+1} = x_i + (t_{i+1} - t_i) f(x_i, u_i) & \quad \forall i \in \{1, \dots, m-1\} & \quad (2.6)
\end{aligned}
$$

Given an initial state $x_{\text{initial}}$ and a control sequence over the time horizon, we use the vehicle model dynamics $f$ and numerical integration to compute the state at each time step. This formulation introduces a coupling constraint $\mathcal{C}$ that governs the relationship between state and control inputs.

One challenge is that this formulation does not yet conform to disciplined convex programming (DCP) principles. We will address this issue by discussing applicable modeling techniques, including approximations and their implications, focusing first on the point mass model.

### Convex Optimization

A set $K \subset \mathbb{R}^n$ is called convex if, for all $x, y \in K$ and $\lambda \in [0, 1]$, the following condition holds

$$\lambda x + (1 - \lambda) y \in K \quad (2.7)$$

A real-valued function $f$ defined over a convex subset $X$ of a vector space is called convex if, for all $x, y \in X$ and $\lambda \in [0, 1]$, the inequality below is satisfied:

$$f(\lambda x + (1 - \lambda) y) \leq \lambda f(x) + (1 - \lambda) f(y) \quad (2.8)$$

A function $f$ is said to be concave if $-f$ is convex.

An optimization problem is defined by a feasible set $X \subset \mathbb{R}^n$ and an objective function $f : X \to \mathbb{R}$. The goal is to find:

$$min_{x \in X} f(x)$$

**Disciplined Convex Programming (DCP)**

For an optimization problem to be efficiently and reliably solvable, it must adhere to the principles of Disciplined Convex Programming (DCP). These rules can be summarized as follows:

An optimization problem $(X, f)$ satisfies the DCP rules if the feasible set $X$ is defined by a series of equality and inequality constraints, where each constraint conforms to one of the following patterns:

- $affine = affine$
- $convex \leq concave$
- $concave \geq convex$

Additionally, the objective function $f$ must be convex. By adhering to these rules, we ensure that state-of-the-art solvers can efficiently find optimal solutions.

To be more precise, each constraint in a convex optimization problem consists of a left-hand side (LHS) and a right-hand side (RHS), both of which can be expressed as functions $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}$. The DCP rules can be interpreted as follows:

- $affine = affine$ as $f, g$ are real-valued affine functions
- $convex \leq concave$ as $f$ is convex over $\mathbb{R}^n$ and $g$ is concave over $\mathbb{R}^n$
- $concave \geq convex$ as $f$ is concave over $\mathbb{R}^n$ and $g$ is convex over $\mathbb{R}^n$

If a problem adheres to these rules, it is considered a convex optimization problem. However, these rules are not comprehensive. It is possible to create valid convex expressions and models that fall outside the scope of these rules. In such cases, additional analysis may be needed to verify convexity.

Advanced solvers, such as 'CVX', apply these rules to systematically determine whether an expression is affine, convex, or concave. For further details on how these solvers handle disciplined convex programming, refer to their documentation: https://web.cvxr.com/cvx/beta/doc/dcp.html

Since solvers cannot operate directly on function spaces, the first step is to approximate the infinite-dimensional space of trajectories using a finite-dimensional vector space.

Additionally, we transform the constraints, originally defined by set membership and predicates, into a system of equalities and inequalities, which are compatible with standard solver inputs.

An alternative approach involves using penalty or barrier functions to incorporate constraints into the objective function.

## 2.2 Vehicle Models

A vehicle model describes the position and orientation of the vehicle in the real world and predicts how these states change over time. Different models vary in complexity and accuracy. Generally, higher complexity results in increased accuracy.

Here, we introduce two fundamental models.

The state variables $x_i$ represent the current position and pose of the vehicle, potentially including velocity, steering angle, or other relevant parameters. We group these variables into a state vector $x$.

The control inputs $u_i$ are external influences that modify the state. These are grouped into a control vector $u$. Both $x$ and $u$ are time-dependent.

### 2.2.1 Point Mass Model

The point mass model (PM) consists of four state variables:

$$x = \begin{bmatrix} p_x \\ p_y \\ v_x \\ v_y \end{bmatrix} \tag{2.9}$$

Here, $p_x$ and $p_y$ represent the vehicle's position in a global fixed coordinate system. Instead of an explicit orientation, velocity is divided into its components $v_x$ and $v_y$.

The model has two control inputs:

$$u = \begin{bmatrix} a_x \\ a_y \end{bmatrix} \tag{2.10}$$

These correspond to accelerations in the $x$ and $y$ directions.

The future position and velocity are determined by the following system of differential equations:

$$\dot{x} = Ax + Bu \tag{2.11}$$

where

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.12}$$

The control inputs are bounded by a circular constraint, where the radius $a_{\text{max}}$ is a model parameter:

$$\sqrt{u_1^2 + u_2^2} \leq a_{\text{max}} \tag{2.13}$$

This model is the simplest commonly used for motion planning.

### 2.2.2 Bicycle Model

The kinematic single track model (KST) consists of five state variables:

$$x = \begin{bmatrix} p_x \\ p_y \\ \delta \\ v \\ \psi \end{bmatrix} \tag{2.14}$$

Similar to the point mass model, the first two state variables $p_x$ and $p_y$ define the vehicle's global position in a two-dimensional coordinate system. The vehicle is now modeled with an orientation $\psi$ relative to the global $x$-axis. The velocity vector describes the velocity $v$ of the rear wheel, aligning with the orientation. The front wheels can rotate around the yaw axis, and their angle relative to the orientation is represented by the steering angle $\delta$.

Two control inputs modify the velocity and steering, directly affecting the state variables:

$$u = \begin{bmatrix} v_\delta \\ a_{\text{long}} \end{bmatrix} \tag{2.15}$$

where $v_\delta$ is the steering velocity, and $a_{\text{long}}$ is the longitudinal acceleration.

The future state follows these differential equations:

$$\dot{p}_x = v\cos(\psi) \tag{2.16}$$
$$\dot{p}_y = v\sin(\psi) \tag{2.17}$$
$$\dot{\delta} = v_\delta \tag{2.18}$$
$$\dot{v} = a_{\text{long}} \tag{2.19}$$
$$\dot{\psi} = \frac{v}{l_{wb}}\tan(\delta) \tag{2.20}$$
$$\tag{2.21}$$

The single-track name originates from simplifying front and rear wheels into single contact points, assuming no wheel slip, leading to a kinematic model abstraction. The following figure comprehends the whole model nicely.

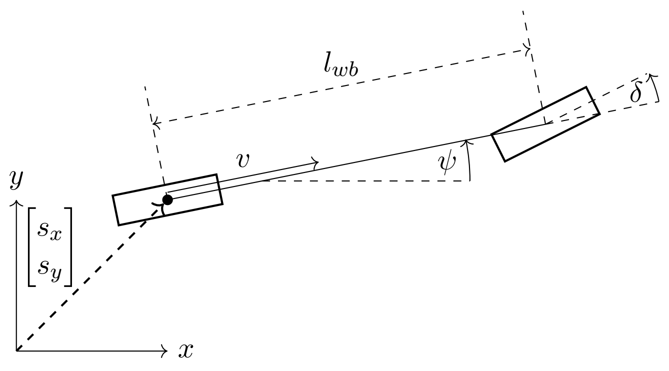


Figure 2.1: Bicycle model representation of a vehicle.

The following additional constraints are part of the model, a parameter $a_{max}$ is introduced:

$$\sqrt{u_2^2 + (x_4\dot{x}_5)^2} \leq a_{\max} \tag{2.22}$$

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

### 2.2.3 Curve Following Coordinate System

# 3 Methodology

## 3.1 Motion Planning Using Point Mass

In Section **??**, we introduced the point mass model with global coordinates and no orientation. We now extend this model to the Frenet frame, which allows for a more structured approach to motion planning along a predefined road.

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

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

Additionally, let $\psi$ be the orientation of the vehicle, and define the alignment error $\xi$ as the difference between the vehicle's orientation and the reference path's tangent angle:

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

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

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

The first-order derivatives are given by:

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

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

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

For the second derivatives, we obtain:

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

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

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

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

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

These expressions incorporate the effects of curvature and alignment, ensuring that motion planning remains accurate and consistent with the road geometry. In subsequent sections, we will leverage these equations to develop motion planning strategies that optimize trajectory feasibility and control performance.

### 3.1.1 Conversion of Global Cartesian Coordinates to Frenet Frame Coordinates

We now define the state variables $x_{tn}$, which represent the vehicle's position, orientation, and velocities in the Frenet frame, and control inputs $u_{tn}$ for our point mass model in the Frenet frame:

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

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

We define the following three control inputs:

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

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

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

With this formulation, we are now able to model vehicle motion in the Frenet frame along a predefined road. This approach facilitates the definition of road topology constraints in a straightforward manner, ensuring that the resulting constraints align with the principles of disciplined convex programming (DCP). However, the inclusion of curvature terms introduces non-convexity into the system dynamics. In the following section, we will explore methods to address this challenge and develop techniques to handle the resulting non-convex constraints effectively.

### 3.1.2 Constraints

First thing we are going to do is that we decouple the input. If you take a look at the dynamics equations 3.11, the last two entries both contain the control input $a_x$, $a_y$ if you plug in the equations 3.7 and 3.8. By making the following Assumption.

#### Assumption: Alignment Error

Orientation of the vehicle $\psi$ equals the angle of the road $\theta$:

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

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

- $[a_x, a_y] = [a_{x,tn}, a_{y,tn}]$
- $\dot{\psi} = \dot{\theta} = \dfrac{d\theta}{ds} \cdot \dfrac{ds}{dt} = C(s)\dot{s}$
- $a_{\psi} = \ddot{\psi} = \ddot{\theta} = C'(s)\dot{s}^2 + C(s)\ddot{s}$

At first glance seems like we are removing the orientation from our model, but we actually just force the vehicle to always be aligned with the road. And since we allow lateral acceleration of the vehicle, it is not fixed to a constant offset to the reference and can still move left or right. Once the trajectory has been planned we let further layers handle the problem of finding the correct control namely steering and longitudinal acceleration for the vehicle, this will be based on more complex models.

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

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

In summary, by fixing one state variable, which leads to slightly less realistic vehicle model, but which led to affine linear dynamics in the controls inputs. This enabled us to introduce artificial variables, which linearizes the dynamics entirely as follows.

### Resulting Simplified Model

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

$$x_{tn} = \begin{bmatrix} s, & n, & \dot{s}, & \dot{n} \end{bmatrix}$$

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

$$\tilde{u} = \begin{bmatrix} u_t, & u_n \end{bmatrix}$$

The dynamics are given by:

$$f(x_{tn}, \tilde{u}) = \begin{bmatrix} \dot{s} \\ \dot{n} \\ u_t \\ u_n \end{bmatrix} \tag{3.14}$$

### Constraints

We have tackled the dynamics constraints of our discrete-time optimal trajectory planning problem 2.6. Next we are going to define our constraints coupling constraints on the states variables and the controls inputs and see which Problem arise and how we can deal with them.

Let us first have a look on the vehicle constraints. Let $\square$ be one of the variables used in trajectory planning, the upper bound is annotated with $\overline{\square}$ and the lower bound with $\underline{\square}$ which are both constant during planning.

Constraints on the velocity is defined in a body fixed manner, with upper and lower bound, where we use the following annotations.

$$\underline{v_x} \leq v_x \leq \overline{v_x} \tag{3.15}$$
$$\underline{v_y} \leq v_y \leq \overline{v_y} \tag{3.16}$$

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

$$\underline{v_x} \leq \dot{s}(1 - nC(s)) \leq \overline{v_x} \tag{3.17}$$
$$\underline{v_y} \leq \dot{n} \leq \overline{v_y} \tag{3.18}$$

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

$$a_x^2 + a_y^2 \leq c \in \mathbb{R}^+ \tag{3.19}$$

for some constant radius $c$, the second similar to the velocity as follows:

$$\underline{a_x} \leq a_x \leq \overline{a_x} \tag{3.20}$$
$$\underline{a_y} \leq a_y \leq \overline{a_y} \tag{3.21}$$

Using our derived equations we can define a mapping from the models state variables and artificial variables to our body fixed accelerations. (TODO: the main paper mentions $a_b = a_{tn}$ which I believe should be

$$a_b + \begin{bmatrix} -\dot{s}(1 - nC(s))C(s)\dot{s} \\ \dot{n}C(s)\dot{s} \end{bmatrix} = a_{tn}$$

)

$$g(x_{tn}, \tilde{u}) := \begin{bmatrix} (1 - nC(s))u_t - (2\dot{n}C(s)\dot{s} + nC'\dot{s}^2) - \dot{s}(1 - nC(s))C(s)\dot{s} \\ u_n + C(s)\dot{s}^2(1 - nC(s)) + \dot{n}C(s)\dot{s} \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \end{bmatrix} \tag{3.22}$$

Combining the constraints on the vehicle derive from the body fixed notion and the straight forward constraint given by the road constraints we can define our coupling constraint set $\mathcal{C}$ as:

$$\mathcal{C} := \left\{ \begin{bmatrix} x_{tn} \\ \tilde{u} \end{bmatrix} \middle| \begin{array}{l} \underline{s} \leq s \leq \overline{s}, \\ \underline{n}(s) \leq n \leq \overline{n}(s), \\ \underline{v_x} \leq \dot{s}(1 - nC(s)) \leq \overline{v_x} \\ \underline{v_y} \leq \dot{n} \leq \overline{v_y} \\ \underline{\dot{\psi}} \leq C(s)\dot{s} \leq \overline{\dot{\psi}}, \\ \underline{a_\psi} \leq C'\dot{s}^2 + C(s)u_t \leq \overline{a_\psi}, \\ \begin{bmatrix} \underline{a_x} \\ \underline{a_y} \end{bmatrix} \leq g(x_{tn}, \tilde{u}) \leq \begin{bmatrix} \overline{a_x} \\ \overline{a_y} \end{bmatrix} \\ ||g(x_{tn}, \tilde{u})||^2 \leq c \end{array} \right\} \tag{3.23}$$

This set is highly non-convex, and we now face the problem to 'convexify' those constraints. We will do so by finding an inner polytope of the set $\mathcal{C}$ stated as follows.

**Problem Definition: Finding an Inner Polytope**

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

$$\underline{\mathcal{C}} = \left\{ \begin{bmatrix} x_{tn} \\ \tilde{u} \end{bmatrix} \middle| N \begin{bmatrix} x_{tn} \\ \tilde{u} \end{bmatrix} \leq b \right\} \subseteq \mathcal{C} \tag{3.24}$$

We will demonstrate two methods how one can archive, both them try to archive the following.

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

$$\underline{\tilde{\mathcal{C}}} = \left\{ \begin{bmatrix} \dot{s} \\ u_t \\ u_n \end{bmatrix} \middle| \begin{bmatrix} x_{tn} \\ \tilde{u} \end{bmatrix} \in \mathcal{C}, \text{for all } \begin{bmatrix} s \\ n \\ \dot{n} \end{bmatrix} \in \begin{bmatrix} \underline{s}, \overline{s} \\ \underline{n}, \overline{n} \\ \underline{\dot{n}}, \overline{\dot{n}} \end{bmatrix} \right\} \tag{3.25}$$

To be more specific, we want to find a set of linear constraints, which each spans a half space and let $\underline{\tilde{\mathcal{C}}}$ be intersection of all those half spaces, then the following should hold.

$$\begin{bmatrix} \dot{s} \\ u_t \\ u_n \end{bmatrix} \in \underline{\tilde{\mathcal{C}}} \implies \forall \begin{bmatrix} s \\ n \\ \dot{n} \end{bmatrix} \in \begin{bmatrix} \underline{s}, \overline{s} \\ \underline{n}, \overline{n} \\ \underline{\dot{n}}, \overline{\dot{n}} \end{bmatrix} : \begin{array}{ll} (\underline{v_x} \leq \dot{s}(1 - nC(s)) \leq \overline{v_x} & \wedge \\ \underline{\dot{\psi}} \leq C(s)\dot{s} \leq \overline{\dot{\psi}} & \wedge \\ \underline{a_\psi} \leq C'\dot{s}^2 + C(s)u_t \leq \overline{a_\psi} & \wedge \\ \begin{bmatrix} \underline{a_x} \\ \underline{a_y} \end{bmatrix} \leq g(x_{tn}, \tilde{u}) \leq \begin{bmatrix} \overline{a_x} \\ \overline{a_y} \end{bmatrix} & \wedge \\ ||g(x_{tn}, \tilde{u})||^2 \leq c) & \end{array} \tag{3.26}$$

The problem therefore reduces to the Elimination of the $\forall$ quantifier.

### 3.1.3 Eliminating the For-All Operator

#### Fitting a Box

The first approach is the computationally less expensive one, but comes in cost with smaller resulting set, but is it turn it does still perform really well. The idea is to find intervals for the variables of interest namely $\dot{s}$, $u_t$ and $u_n$ such that for the values variables may take from those intervals the implications from 3.26 holds. Each of the condition of that implications follow the following pattern. Let $x \in \{\dot{s}, u_t, u_n\}$ and $y$ a vector containing the remaining variables are part of the condition.

$$c_{min} \leq f(x, y) \leq c_{max} \tag{3.27}$$

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

$$f(x, y) = a(y)x + b(y) \tag{3.28}$$

with $a, b : \mathbb{R}^n \to \mathbb{R}$, since $a$ and $b$ are continues functions over a bounded domain $Y$ one can find bounds on $a(y)$ and $b(y)$:

$$a_{min} \leq a(y) \leq a_{max}, \quad b_{min} \leq b(y) \leq b_{max} \tag{3.29}$$

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

$$x \in [\underline{x}, \overline{x}] \implies \forall y \in Y : c_{min} \leq f(x, y) \leq c_{max} \tag{3.30}$$

We define $X := [\underline{x}, \overline{x}]$. We can calculate $X$ with a case distinction over the possible signs $a(y)$ may take. Let's start with

**1.** $a(y) > 0$: We can subtract 3.27 with $b(y)$ and divide by $a(y)$:

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

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

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

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

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

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

if it does not hold $X$ is given by $X = \emptyset$, and otherwise we exclude all $y \in Y$ for which $a(y) = 0$ and go to the first case.

**3.** $a(y) < 0$: We can again subtract $b(y)$ from 3.27 and divide by $a(y)$, but the directions of the inequalities changes this time:

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

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

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

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

**4.** $a(y) \leq 0$: similar to the second case we have to if $c_{min} \leq b_{min}$ and $b_{max} \leq c_{max}$ holds, if not $X = \emptyset$ else we ignore the values for $a(y)$ takes the value zero and go to third case.

**5.** We have so far considered all the cases where $a(x)$ can not take both positive and negative values, we remain with the last case, where $a_{min} < 0$ and $0 < a_{max}$. Since $a(y) = 0$ for some values we check $c_{min} \leq b_{min}$ and $b_{max} \leq c_{max}$ and set $X = \emptyset$ if it does not hold, 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, since one can go step by step through the constraints and apply these rules. Of course the resulting Polytope is of a box shape based on the interval approach and therefore reduces our set of possible state variables and control inputs further.

**Evaluation of the Inner Polytope**

We start by defining the intervals for the state variables and control inputs:

$$\begin{aligned} 0 &\leq s \leq 10, \\ 0 &\leq n \leq 2, \\ -2 &\leq \dot{n} \leq 2 \end{aligned}$$

We set $C(s) = \dfrac{1}{400}$ constant. Next, we plug in concrete values for the upper and lower bounds of the constraints:

$$\begin{aligned} 0 &\leq \dot{s}(1 - nC(s)) \leq 10, \\ -5 &\leq C(s)\dot{s} \leq 5, \\ -2 &\leq C'(s)\dot{s}^2 + C(s)u_t \leq 2, \\ \begin{bmatrix} -3 \\ -4 \end{bmatrix} &\leq g(x_{tn}, \tilde{u}) \leq \begin{bmatrix} 6 \\ 4 \end{bmatrix}. \end{aligned}$$

Applying the approach outlined above, we can determine the intervals for $\dot{s}$, $u_t$, and $u_n$ that satisfy the constraints.

$$\begin{aligned} 0 &\leq \dot{s} \leq 10 \\ -2.9 &\leq u_t \leq 5.9 \\ -3.99 &\leq u_n \leq 3.75 \end{aligned}$$

**Cylindrical Algebraic Decomposition**

The second approach is to use Cylindrical Algebraic Decomposition (CAD) to find the inner polytope. This method is computationally more expensive but provides a more accurate result. The idea is to find for a given formula (containing quantifier) an equivalent formula without quantifiers, which will contain only the free variables that are not bound to a quantifier. This can be done by using CAD.

CAD is a method used in computer algebra for solving systems of polynomial equations and inequalities. Let's say you have a set of polynomial equations and inequalities. If one applies CAD to this set, it will decompose the space into a finite number cylindrical cells. Each cell is described by a sequence of polynomial inequalities. The cells have the property that they have a constant truth value over the input set of polynomial equations and inequalities. This way one has to only pick one point from each cell to check the truth value of the input set of polynomial equations and inequalities. The number of cells grows doubly exponentially with the number of variables in the input set of polynomial equations and inequalities. For CAD exist several implementations.

We are going to illustrate how the Algorithm works and it is not scope of this work to explain the implementations. Instead, we are giving you a toolbox how one can do it.

**Example**

Since we are focusing on eliminating $\forall$ quantifiers, we will tackle the following problem:

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

Since the Quantifier Elimination problem is usually done the on an existential quantifier, we first solve the problem for

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

and once we have the solution for this problem we can take the compliment over $\mathbb{R}$ to get the solution for the original problem. The first thing is to extract is to apply CAD to the polynomial $x^2 + bx + 1$, which results in 7 cells illustrated in the figure 3.1. Most of the cells are open and if the edge of the cell is part of it then it is depicted dashed with the same color.

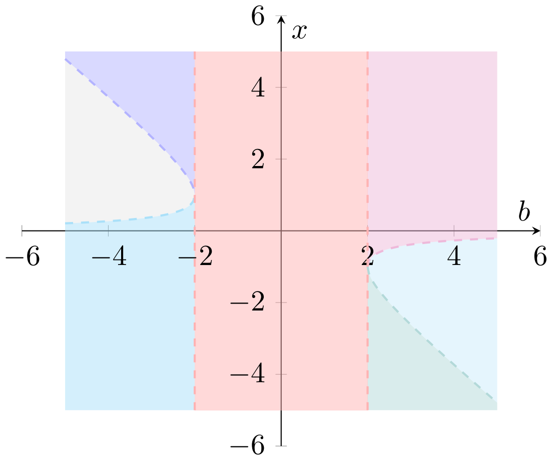


Figure 3.1: Illustrating the cells with shaded regions.

We now have to check the truth value of the polynomial inequality $x^2 + bx + 1 < 0$ for each cell, by picking one random sample and evaluating the inequality. The cells with a truth value of true colored in green then get project the $b$-axis. The resulting intervals are the result of the existential quantifier elimination $(-\infty, -2) \cup (2, \infty)$. As an input of the Algorithm one has to define an order of precedence for the variables. The algorithm in its first phase projects the space iterative 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 which is last in the order of precedence. Therefor if one defines $b$ as the first variable, the sets of polynomial inequalities of each cell will always contain an interval for $b$, which is the corresponding projection on the axis. This allows to just read the resulting intervals.

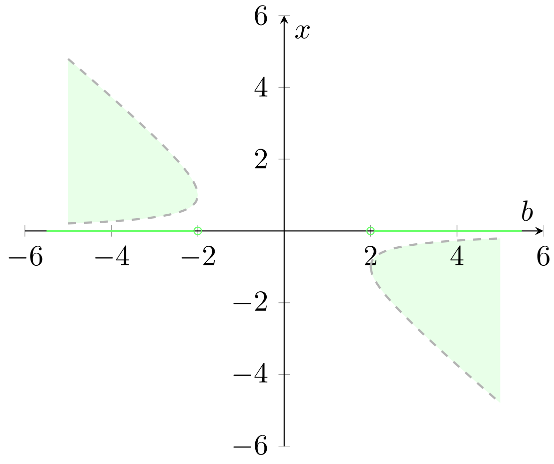


Figure 3.2: Illustrating the remaining cells.

Since $(-\infty, -2) \cup (2, \infty)$ is an equivalent formula for $\exists x, x^2 + bx + 1 < 0$, the solution for the original problem is given by negotiate both formulas.

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

Using this technique for eliminating quantifiers of formulas, we can apply it to our of finding an inner polytope.

### Compare both Approaches

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

- $C(s) \in \left[-\frac{1}{200}, 0\right]$
- $s \in [0, 10]$
- $n \in [0, 2]$
- $v_x \in [0, 10]$
- $v_y \in [-2, 2]$

- $a_x \in [-3, 6]$
- $a_y \in [-4, 4]$
- $\dot{\psi} \in [-5, 5]$
- $a_\psi \in [-2, 2]$

The results are shown in the following figures.

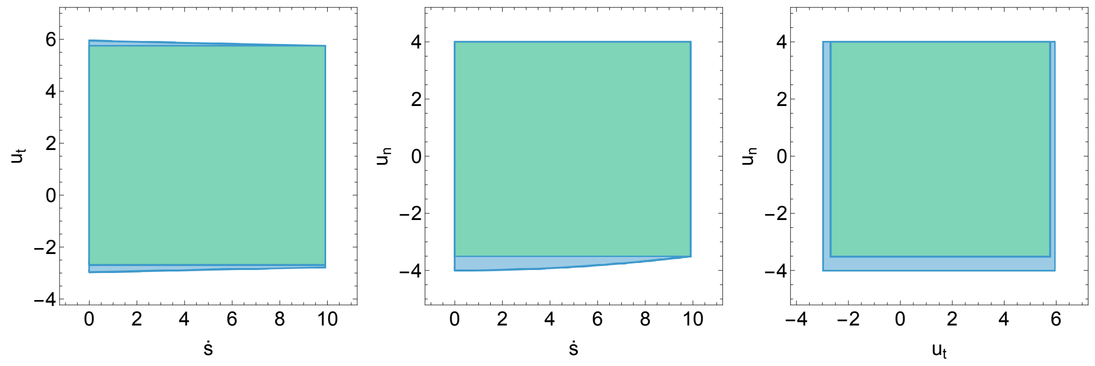


Figure 3.3: In Green our first approach and in Blue using CAD.

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

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

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

### Inner Polytope

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

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

$$\underline{\mathcal{C}} = \underline{\tilde{\mathcal{C}}} \times [\underline{s}, \overline{s}] \times [\underline{n}, \overline{n}] \times [\underline{\dot{n}}, \overline{\dot{n}}] \tag{3.31}$$

with $\underline{\tilde{\mathcal{C}}}$ resulting from $\forall$-elimination.

### Limitations and Outlook

While the $\forall$-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 method relies on the assumption that the function $f(x, y)$ is affine in $x$. If this assumption does not hold, the approach may not be applicable or may yield inaccurate results.
- **Worst-Case Scenarios:** The intervals are determined based on worst-case scenarios, which can be overly pessimistic and exclude feasible solutions that are not captured by the worst-case analysis.

Overall, while the $\forall$-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.

Imagine your scenario consists of a tight turn and long straight road. In such scenarios, the model, will restrict $\dot{s}$ to an interval that is valid for both the tight turn and the straight road. One can easily imagine that model will find a result, but it will not be able to drive fast on the straight road, then it is possible to drive on tight the turn.

Solving such issues can be done by introducing 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. But this leads to a new Problem, how to switch between the segments. We have come up with a solution where we use the current vehicle velocity to predict at which time the vehicle will reach the end of the segment. Since one knows the vehicles current position, velocity and the distance to the end of the segment, one can calculate the time it will take to reach.

What both approaches both do not consider and may lead to a far larger set, if you would restrict $\dot{n}$ for example 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 then is used for $u_t$ and finally for $u_n$. But changing the order may lead to better results for desired driving behavior.

Since the first approach is so simple we implemented a non-linear program which defines the relations between the intervals that arise with variables as upper and lower bounds. Constraining them additional we can define an objective that can model certain driving behavior. As for example one should be able to slow down as quickly as possible, or the upper speed limit should as large as possible. The latter one leads for example 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 could be considered better, since one only loses a bit off longitudinal acceleration, but therefore can driver faster.

But nevertheless, we managed to build our first vehicle model for motion planning, which can be solved by convex solver. We will now increase the complexity for the next model, in which we will tackle the problem of introducing a vehicle orientation and a steering angle during the planning. This will allow us to get a more realistic trajectory for the vehicle, especially for lower speeds and turns.

### 3.1.4 Determining the Steering Angle

Given: A car model which has state:

$$x_{car} = \begin{bmatrix} p_x \\ p_y \\ \delta \\ v \\ \psi \\ \dot{\psi} \\ \beta \end{bmatrix}$$

$p_x$ Global Position, $p_y$ Global Postion, $\delta$ Steering Angle, $v$ Velocity, $\psi$ Orientation, $\dot{\psi}$ Yaw Rate, $\beta$ Slip Angle.
and control inputs:

$$u_{car} = \begin{bmatrix} \dot{\delta} \\ a \end{bmatrix}$$

$\dot{\delta}$ Steering Angle Rate, $a$ Longitudinal Acceleration

The used planning model has its own state definition $x_{planning}$ and control inputs $u_{planning}$.

A planning model has to provide a mapping from the car states to its states:

$$x_{car} \mapsto x_{planning}$$

and a mapping from its control inputs to the control inputs of the car:

$$u_{planning} \mapsto u_{car}$$

$$x_{planning} = \begin{bmatrix} s \\ n \\ \dot{s} \\ \dot{n} \end{bmatrix}$$

($s$, $n$) Frenet Frame Coordinates along the road, ($\dot{s}$, $\dot{n}$) Change of the Frenet Frame Coordinates along the road,

$$u_{planning} = \begin{bmatrix} u_t \\ u_n \end{bmatrix}$$

$u_t$, $u_n$ artificial control inputs, which can be mapped ($a_s$, $a_n$) Acceleration of the Frenet Frame Coordinates along the road using $g$.

$$x_{car} = \begin{bmatrix} p_x \\ p_y \\ \delta \\ v \\ \psi \\ \dot{\psi} \\ \beta \end{bmatrix} \mapsto \begin{bmatrix} self.road.get\_road\_position(x,y)[0] \\ self.road.get\_road\_position(x,y)[1] \\ v\cos(\psi - \theta(s)) \\ v\sin(\psi - \theta(s)) \end{bmatrix}$$

For control input we additional require the current state $x = [s, n, \dot{s}, \dot{n}]^T$ of the planning model and we need to store the current steering angle of the car $\delta_{cur}$:

**Approach 1**

$$\xi = \arctan(\frac{\dot{n}}{\dot{s}(1 - nC(s))})$$

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

$$\dot{\psi} = \frac{u_n - \tan(\xi)u_t}{v(\tan(\xi)\sin(\xi) + \cos(\xi))}$$

$$a = \frac{u_t + v\dot{\psi}\sin(\xi)}{\cos(\xi)}$$

$$\dot{C} = \dot{C}(s) = C'(s)\dot{s} \approx \frac{C(s + \dot{s}\Delta t) - C(s)}{\Delta t}$$

$$\dot{\xi} = \frac{1}{1 + (\frac{\dot{n}}{\dot{s}(1-nC(s))})^2} \frac{u_n\dot{s}(1 - nC(s)) - \dot{n}(u_t - C(s)(u_t n + \dot{s}\dot{n}) - \dot{C}\dot{s}n)}{(\dot{s}(1 - nC(s)))^2}$$

$$\delta = \arctan((\dot{\xi} + C(s)\dot{s})\frac{l_{wb}}{v})$$

$$v_\delta = \max(\min(\frac{\delta - \delta_{cur}}{\Delta t}, \overline{v_\delta}), \underline{v_\delta})$$

$$u_{planning}, x_{planning} \mapsto \begin{bmatrix} v_\delta \\ a \end{bmatrix} = \begin{bmatrix} \dot{\delta} \\ a \end{bmatrix}$$

**Approach 2**

$$[s_0, n_0, \dot{s}_0, \dot{n}_0]^T := [s, n, \dot{s}, \dot{n}]^T$$

$$[s_1, n_1, \dot{s}_1, \dot{n}_1]^T := \begin{bmatrix} s_0 \\ n_0 \\ \dot{s}_0 \\ \dot{n}_0 \end{bmatrix} + \Delta t \begin{bmatrix} \dot{s}_0 \\ \dot{n}_0 \\ u_n \\ u_t \end{bmatrix}$$

$$v_0 = \sqrt{(}\dot{s}_0^2 + \dot{n}_0^2)$$

$$v_1 = \sqrt{(}\dot{s}_1^2 + \dot{n}_1^2)$$

$$a = \frac{v_1 - v_0}{\Delta t}$$

$$v_\delta = \arctan(l_{wb}(planned_psi_2 - (cur_psi + 1/l_wb * v_0 * np.tan(cur_steering_angle) * dt))/(v_1 * dt))/dt$$

$$x_1 = \begin{bmatrix} s_1 \\ n_1 \\ \dot{s}_1 \\ \dot{n}_1 \end{bmatrix} = x_0 + \begin{bmatrix} \dot{s}_0 \\ \dot{n}_0 \\ u_t \\ u_n \end{bmatrix} dt$$

$$\begin{bmatrix} a_s \\ a_n \end{bmatrix} = g(u_t, u_n)$$

$$\xi_0 = \arctan\left(\frac{\dot{n}_0}{\dot{s}_0}\right)$$
$$\xi_1 = \arctan\left(\frac{\dot{n}_1}{\dot{s}_1}\right)$$

$$v_0 = \sqrt{\dot{s}_0^2 + \dot{n}_0^2}$$
$$v_1 = \sqrt{\dot{s}_1^2 + \dot{n}_1^2}$$

$$\psi_0 = \xi_0 + \theta(s_0)$$
$$\psi_1 = \xi_1 + \theta(s_1)$$

$$\dot{\psi} = \frac{\psi_1 - \psi_0}{dt}$$

$$u_{planning}, x_{planning} \mapsto \frac{1}{dt} \begin{bmatrix} \arctan(l_{wb}\frac{\dot{\psi}}{v_1}) - \delta_{cur} \\ v_1 - v_0 \end{bmatrix} = \begin{bmatrix} \dot{\delta} \\ a \end{bmatrix}$$

### 3.1.5 Final Model

## 3.2 Motion Planning using Bicycle Model

We have already introduced the kinematic single track model in which a steering angle and an orientation, which will now combine with the idea of the previous chapter. We want to model our state variables in the road aligned frame, which is the Frenet frame. The states and controls variables of the kinematic single track model are defined in the global coordinate system are defined as in 2.14 and 2.15:

Our goal is not model dynamics and state variables in the Frenet frame.

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

(a) State Variables

$$u = \begin{bmatrix} v_\delta \\ a_{\text{long}} \end{bmatrix}$$

(b) Control Inputs

### 3.2.1 Transforming Global Cartesian Coordinates to Frenet Frame

In this section, we derive the state transition model for the Frenet frame, which is a commonly used coordinate system in vehicle dynamics and control. The Frenet frame is particularly useful for path tracking and motion planning as it simplifies the representation of the vehicle's motion relative to a reference path.

We start by considering the dynamics of the kinematic single track model in the global coordinate system. This model captures the essential behavior of a vehicle by representing it as a single point mass with a front and rear axle. The state variables include the position coordinates $(p_x, p_y)$, the orientation angle $\psi$, the steering angle $\delta$, and the longitudinal velocity $v$. The control inputs are the longitudinal acceleration $a_{\text{long}}$ and the steering rate $v_\delta$.

The dynamics of the kinematic single track model in the global coordinate system are:

$$\begin{aligned}
\dot{p}_x &= v\cos(\psi) \\
\dot{p}_y &= v\sin(\psi) \\
\dot{\delta} &= v_\delta \\
\dot{v} &= a_{\text{long}} \\
\dot{\psi} &= \frac{v}{l_{wb}}\tan(\delta)
\end{aligned}$$

Building on the orientation model from the previous chapter, we can extend the point mass model to include body-fixed control inputs. This approach helps us achieve our goal.

Using the previously derived equations 3.3 and 3.4, and incorporating the definitions $\xi = \psi - \theta$ and $\dot{\theta} = \frac{d\theta}{ds}\frac{ds}{dt} = C(s)\dot{s}$, we obtain the following equations:

$$\begin{aligned}\dot{s} &= \frac{v_x \cos\xi}{1 - nC(s)}\\ \dot{n} &= v_x \sin\xi\\ \dot{\xi} &= \dot{\psi} - C(s)\dot{s}\end{aligned}$$

Combining these equations with the kinematic single track model, we derive the state transition model for the Frenet frame. This model provides a comprehensive representation of the vehicle's motion in the Frenet coordinate system, which is essential for accurate path tracking and motion planning.

Next, we define the state variables and control inputs for the single-track model.

### 3.2.2 Model Dynamics Approximation

The state variables and control inputs for the single-track model in the Frenet frame are defined as follows:

$$x = \begin{bmatrix} s \\ n \\ \xi \\ v \\ \delta \end{bmatrix}$$

(a) State Variables

$$u = \begin{bmatrix} a_{x,b} \\ v_\delta \end{bmatrix}$$

(b) Control Inputs

where the state consists of the Frenet frame coordinates $(s, n)$, the heading alignment error $\xi$, the longitudinal vehicle velocity $v$, and the steering angle $\delta$. The state evolution is described by the following equations:

$$f(x,u) = \begin{bmatrix} \dfrac{v\cos\xi}{1 - nC(s)} \\ v\sin\xi \\ \dfrac{1}{l_{wb}} v \tan\delta - C(s)\dot{s} \\ a_{x,b} \\ v_\delta \end{bmatrix}. \tag{3.32}$$

For this model, we will use body-fixed control inputs, which provide convex road topology constraints. To linearize the model dynamics, we will use 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, the following assumptions are made: $nC(s)$ is close to zero. This is a valid since $n$ models 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 be valid. We split up the problem by looking the terms which make the model non-linear.

### Non-Linear Terms

We have four non-linear terms in the state transition model. In the following, we linearize them using approximations.

$$\frac{v\cos\xi}{1-nC(s)} \tag{3.33}$$

$$v\sin\xi \tag{3.34}$$

$$v\tan\delta \tag{3.35}$$

$$C(s)\dot{s} \tag{3.36}$$

With our assumptions, that $nC(s) \approx 0$, we can immediately simplify the first term.

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

### First Order Taylor Polynomial

To linearize the non-linear terms, we use the first order Taylor polynomial approximation around a reference point. The first order Taylor expansion of a function $f(x)$ around a point $x_0$ is given by:

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

Applying the first order Taylor polynomial to the trigonometric functions sin, cos, and tan around the reference points $\xi_0$ and $\delta_0$, we obtain the following approximations:

$$\cos(\xi) \approx \cos(\xi_0) - \sin(\xi_0)(\xi-\xi_0)$$

$$\sin(\xi) \approx \sin(\xi_0) + \cos(\xi_0)(\xi-\xi_0)$$

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

These approximations are known as small angle approximations, which are valid when the angles $\xi$ and $\delta$ are small. In vehicle dynamics, it is often reasonable to assume that the heading alignment error $\xi$ and the steering angle $\delta$ are small, especially when the vehicle is closely following a reference path. This allows us to simplify the trigonometric functions using their first order Taylor expansions.

Substituting these first order Taylor approximations into the state transition model, we obtain the following terms:

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

Since our reference values $\xi_0 = 0$ and $\delta_0 = 0$ are constant the only remaining non-linear terms are:

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

Bilinear terms arise when the product of two variables appears in the equations, making the system non-linear. In our state transition model, the terms $v\xi$, $v\delta$, and $C(s)\dot{s}$ are bilinear. Since $C(s)$ is a function of $s$, we will introduce another assumption. We have already seen that we can model our road in segments, which we used for the Point Mass Model to obtain less conservative coupling constraints. Here, we will reuse this approach to linearize our dynamics.

### Assumption: Piece Wise Linear Curvature

We assume that the curvature can be expressed 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 reduces the non-linear term $C(s)\dot{s}$ to a new bilinear term $s\dot{s}$, which we need to linearize.

Next, we will focus on linearizing the bilinear terms.

### 3.2.3 Convex Relaxation of Bilinear Terms

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

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

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

We introduce a new auxiliary variable $w$, which will be used as an approximation of the bilinear term $xy$. We use linear constraints to bound the auxiliary variable $w$ and ensure that it approximates the bilinear term $xy$. The linear constraints are derived from the bounds of the variables $x$, $y$, and the bilinear term $xy$, as shown below:

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

The idea behind these constraints is to create a convex lower bound and a concave upper bound for the bilinear term $xy$. These bounds are constructed as follows:

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

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

Since a and b are both positive, we can multiply them to get:

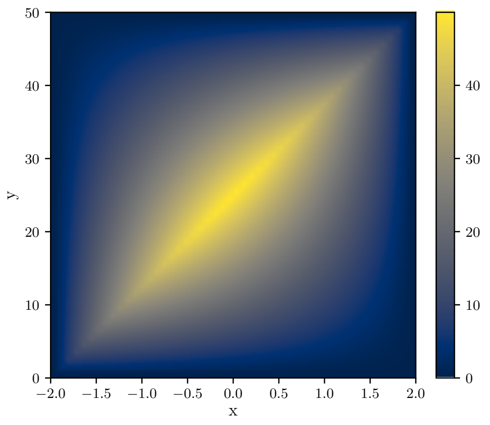
$$ab = xy - \underline{x}y - \underline{y}x + \underline{x}\underline{y} \geq 0$$

Thus, we get our first under estimator for $xy$:

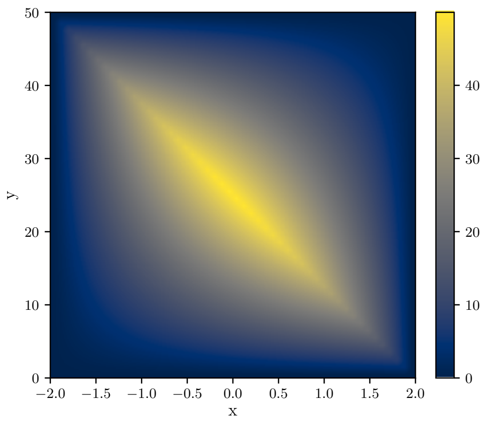
$$xy \geq \underline{x}y + \underline{y}x - \underline{x}\underline{y}$$

To get the all the possible lower and upper bounds by what is given we can apply this pattern to $a \in \{x - \underline{x}, \overline{x} - x\}$ and $b \in \{y - \underline{y}, \overline{y} - y\}$. For any of the four possible combination of $a$ and $b$ we get $ab \geq 0$ and can therefor come up with either an upper or lower bound for $xy$.

Figure 3.7a illustrates the deviation between the actual bilinear term $xy$ and the smallest upper bound provided by the relaxation constraints. Similarly, Figure 3.7b



(a) Difference to the upper bound



(b) Difference to the lower bound

Figure 3.7: McCormick relaxation bounds for the bilinear term $xy$.

shows the deviation to the greatest lower bound. For the range $-2 \leq x \leq 2$ and $0 \leq y \leq 50$. It is evident that the bounds improve as $x$ and $y$ approach their respective limits.

Figures 3.8a and 3.8b present the results when $x$ is more tightly bounded, specifically $-2 \leq x \leq 0$. One can observe that the maximum deviation is considerably reduced compared to the previous scenario, indicating that tighter bounds yield a more accurate relaxation.

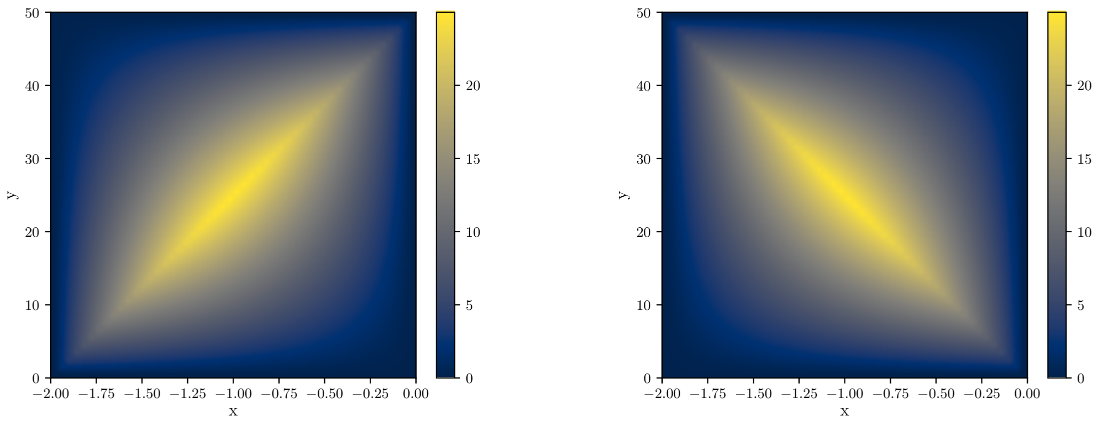
### Improve Bounds

To improve the bounds of the McCormick relaxation, we can use tighter bounds for the variables involved in the bilinear terms. By reducing the range of the variables, we can achieve a more accurate approximation of the bilinear terms. This can be done by analyzing the specific problem and determining the realistic bounds for the variables.

For example, if we know that the variable $x$ is always within a smaller range, such as $-1 \leq x \leq 1$, we can use these tighter bounds to improve the McCormick relaxation. Similarly, if the variable $y$ is within a smaller range, such as $0 \leq y \leq 25$, we can use these bounds to achieve a more accurate approximation.

To apply tighter bounds to the velocity, we can use the known current velocity and the maximum and minimum acceleration values. This allows us to provide a more accurate range for the velocity at each time point, especially for the first few time points.

Let $v_0$ be the current velocity, $\underline{a}$ be the minimum acceleration, and $\overline{a}$ be the maximum acceleration. The velocity at the next time point $v_1$ can be bounded as follows:



(a) Difference to the upper bound

(b) Difference to the lower bound

Figure 3.8: McCormick relaxation bounds for the bilinear term $xy$ with stricter bounds on $x$.

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

where $\Delta t$ is the time step.

For subsequent time points, we can iteratively apply these bounds to obtain tighter ranges for the velocity. For example, the velocity at the second time point $v_2$ can be bounded as:

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

By applying these tighter bounds, we can improve the accuracy of the McCormick relaxation for the bilinear terms involving the velocity. This results in a more precise and efficient representation of the vehicle's motion, which is essential for accurate path tracking and motion planning.

In summary, improving the bounds of the McCormick relaxation involves: 1. Analyzing the specific problem to determine realistic bounds for the variables. 2. Using tighter bounds for the variables involved in the bilinear terms. 3. Applying these tighter bounds to achieve a more accurate approximation of the bilinear terms.

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

### 3.2.4 Final Model

The final linearized state transition model for the Frenet frame, incorporating the linearized non-linear terms and the McCormick relaxation for the bilinear terms, is given by:

$$f(s, n, \xi, v, \delta, a_{x,b}, v_\delta) \approx \begin{bmatrix} v(\cos(\xi_0) + \sin(\xi_0)\xi_0) - \sin(\xi_0) w_{v,\xi} \\ v(\sin(\xi_0) - \cos(\xi_0)\xi_0) + \cos(\xi_0) w_{v,\xi} \\ \frac{v}{l_{wb}}(\tan(\delta_0) - \frac{\delta_0}{\cos^2(\delta_0)}) + \frac{w_{v,\delta}}{\cos^2(\delta_0)} - a_{i(s)} w_{s,\dot{s}} - b_{i(s)}\dot{s} \\ a_{x,b} \\ v_\delta \end{bmatrix} \tag{3.37}$$

Where $w_{v,\xi}$, $w_{v,\delta}$, and $w_{s,\dot{s}}$ are auxiliary variables introduced by the McCormick relaxation to approximate the bilinear terms $v\xi$, $v\delta$, and $s\dot{s}$, respectively. The Curvature is treated as a piece-wise linear function, where $a_{i(s)}$ and $b_{i(s)}$ are the slope and intercept of the linear function in the interval $[s_{i-1}, s_i]$, which can be select by the position $s$. The index will be treated as a constant during planing for each time point, which we can achieve by predicted the road segment at each time point. The additional constraints for the McCormick relaxation are:

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

Figure 3.9: McCormick relaxation constraints for the bilinear terms.

#### Coupling Constraints

The coupling constraints are simple as

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

$$v_\delta \in [\underline{v_\delta}, \overline{v_\delta}], a_{x,b} \in [\underline{a}_{x,b}, \overline{a}_{x,b} \min\{1, \frac{v_S}{v}\}]$$

where $v_S$ is parameter which is used to encounter limiting engine power and breaking power.

Last but not least we have to consider the friction circle.

$$\sqrt{a_{x,b}^2 + (\frac{v^2}{l_{wb}} \tan(\delta))^2} \leq a_{max}$$

Utilizing that $\tan(\delta) \leq \frac{\tan(\overline{\delta})}{\overline{\delta}} \delta$ leads to a bit more conservative constraint.

$$a_{x,b}^2 + (\frac{1}{l_{wb}} \frac{\tan(\overline{\delta})}{\overline{\delta}})^2 v^4 \delta^2 \leq a_{max}^2$$

Next, we derive an upper bound for the term $v^4\delta^2$. This term appears in the friction circle constraint and can significantly impact the vehicle's dynamics. By finding an upper bound, we can simplify the constraint and ensure that the model remains computationally efficient while maintaining accuracy.

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

Our goal is to find the parameters for the following term:

$$av^2 + b\delta^2$$

Such that this term equals the term $v^4\delta^2$ for the absolute maximum values of $v$ and $\delta$ $v^* = max\{|\underline{v}|, |\overline{v}|\}$, $\delta^* = max\{|\underline{\delta}|, |\overline{\delta}|\}$.

We end up with the following values:

$$a = max\left\{\frac{(v^*)^4(\delta^*)^2 - \frac{(\delta^*)^3}{v^*+\delta^*}}{(v^*)^2}, 0\right\}$$

$$b = \begin{cases} (v^*)^4 & \text{if } a = 0 \\ \dfrac{\delta^*}{v^* + \delta^*} & \text{otherwise} \end{cases}$$

We can use this upper bound to enforce the friction circle by the following constraint:

$$a_{x,b}^2 + (\frac{1}{l_{wb}} \frac{\tan(\overline{\delta})}{\overline{\delta}})^2 (av^2 + b\delta^2) \leq a_{max}^2$$

Since $a_{x,b}$, $v$, and $\delta$ are the only variables in this constraint and each of them only occur squared, without any interaction, we have a convex constraint which we can use to enforce the friction circle.

## 3.3 Modeling Techniques

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

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

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

### 3.3.1 Soft Constraints

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

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

Consider a constraint of the form $g(x) \leq 0$. To make this constraint soft, we introduce a slack variable $s \geq 0$ and modify the constraint to $g(x) \leq s$. We then add a penalty term $\lambda s$ to the objective function, where $\lambda$ 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 $\lambda$, we can control the degree to which the soft constraint is enforced. A larger $\lambda$ places more emphasis on satisfying the constraint, while a smaller $\lambda$ allows for greater flexibility in violating the constraint.

### 3.3.2 Auxiliary Variables

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

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

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.

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

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:

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

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

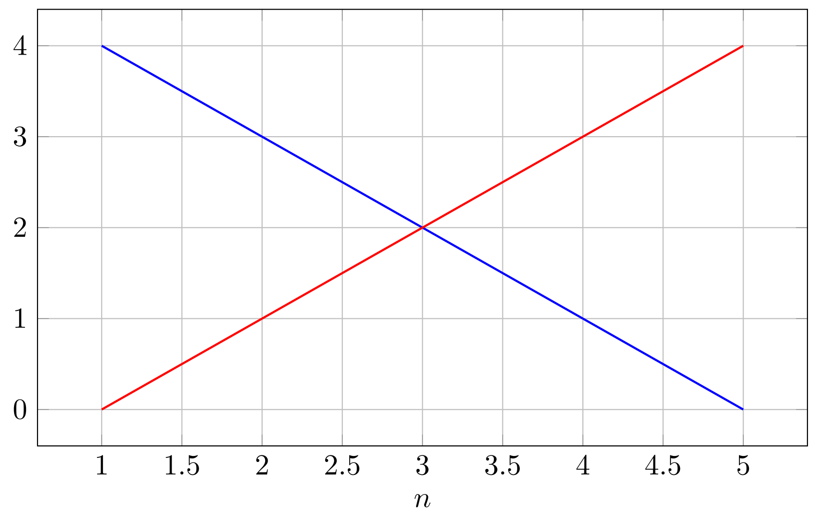


Figure 3.10: Plot of distance to road boundaries

In this plot 3.10, the blue line represents $\overline{n}(s) - n$, the red line represents $n - \underline{n}(s)$. We can also plot the objective function for the new formulation.

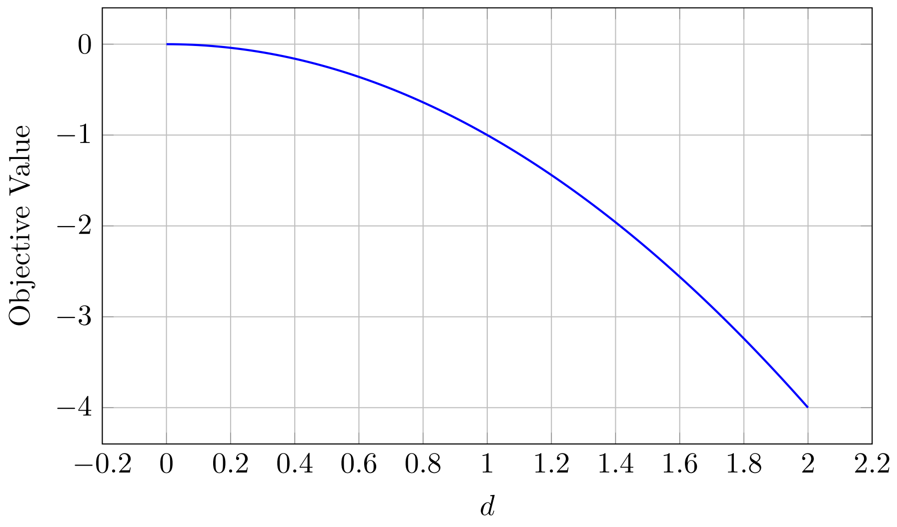


Figure 3.11: Plot of the objective function

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

### 3.3.3 Penalty Methods

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

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

The modified optimization problem can be written as:

$$\min_x \quad f(x) + \mu \max(0, g(x))^2$$

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

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

### 3.3.4 Lagrangian Relaxation

Lagrangian relaxation is a technique used to simplify complex optimization problems by relaxing some of the constraints and incorporating them into the objective function using Lagrange multipliers. This approach transforms the original problem into a simpler one that can be solved more easily.

Consider an optimization problem with a constraint $g(x) \leq 0$. In Lagrangian relaxation, we introduce a Lagrange multiplier $\lambda \geq 0$ and form the Lagrangian function:

$$L(x, \lambda) = f(x) + \lambda g(x)$$

The relaxed optimization problem can be written as:

$$\begin{aligned} \min_x \quad & L(x, \lambda) \\ \text{subject to} \quad & \lambda \geq 0 \end{aligned}$$

By solving the relaxed problem for different values of $\lambda$, we can obtain a lower bound on the optimal value of the original problem. The quality of the bound depends on the choice of $\lambda$, and finding the best value of $\lambda$ is an optimization problem in itself.

Lagrangian relaxation is particularly useful in combinatorial optimization problems, where it can provide strong bounds and guide the search for optimal solutions. It is also a key component of more advanced techniques, such as the Lagrangian duality and the subgradient method.

### 3.3.5 Conclusion

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

# 4 Evaluation

In this chapter, we evaluate the performance of our proposed method on a set of benchmark problems. We introduced two models, each with its own dynamic equations and coupling constraints on state variables and control inputs. For the Point Mass Model, the dynamics are given by 3.14 and its coupling constraints by 3.31, which can be constructed using the two proposed methods for eliminating the for-all operator. The dynamics of the kinematic single track model are approximated by 3.37, and the coupling constraints are listed in section 3.2.4.

For realistic driving tests, we still need to address implementation details, which we will handle next. Additionally, we require a simulation model and a control layer implementation to map the planned trajectory to the vehicle. The simulation model is used to evaluate the performance of the planned trajectory, while the control layer maps the planned trajectory to the vehicle. This allows us to construct actual scenarios and test our models in driving conditions.

## 4.1 Implementation Details

### 4.1.1 Road Segments

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

Our planner operates on a time horizon, divided into discrete time steps $\{t_i\}_{i=1,\dots,n}$. 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:

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

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

Our trajectory planner requires the model $(\dim(x), \dim(u), f, \mathcal{C})$, 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 $t_i$:

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

For each $i \in \{1,\dots,n\}$ we define an equality constraint:

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

Those combined with the coupling constraints $\mathcal{C}$ and the additional constraints for the kinematic single track model 3.9, define the constraints for our discrete time optimal trajectory problem.

With the constant state $x_{initial}$, we will model our initial state and add the following constraint $x(t_0) = x_{initial}$. For our evolution we did not encounter any additional constraints on the final state. Instead, we modeled our driving behavior through the objective.

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

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

## 4.2 Performance Evaluation

### 4.2.1 Objectives

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

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

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

where $d_1(t_i) \in \mathbb{R}^{dim(u)}$ is a new introduced auxiliary variable, which is constrained by:

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

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

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

where $d_2(t_i) \in \mathbb{R}$ is a new introduced auxiliary variable, which is constrained by:

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

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

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

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

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

where $\alpha$, $\beta$, and $\gamma$ 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, adherence to the reference trajectory, and reaching the desired terminal state.

### 4.2.2 Scenarios

In order to evaluate the performance of our trajectory planner, we implemented several driving scenarios. These scenarios are designed to test different aspects of the planner's capabilities. The scenarios we implemented are:

The Straight Road scenario tests 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 evaluated based on its ability to execute a smooth left turn while adhering closely to the reference trajectory.

The Lane Change scenario assesses 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 can be visualized as follows:

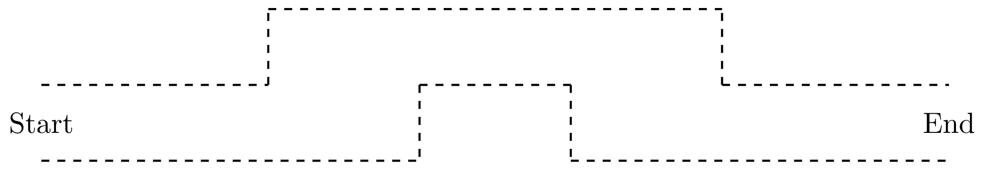


Figure 4.1: Elchtest scenario visualization

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

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

### 4.2.3 Simulation

For the actual simulation of the vehicle, we use a complex model and discretized its dynamics using Runge-Kutta which provides a more accurate than the forward we used during planning.

For reproducibility, we provide the model which is given by the state variables and control inputs:

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

where $p_x, p_y$ are the position coordinates, $\delta$ is the steering angle, $v$ is the velocity, $\psi$ is the yaw angle, $\dot{\psi}$ is the yaw rate, $\beta$ is the slip angle, $a_x$ is the longitudinal acceleration, and $v_\delta$ is the steering rate.

The dynamics of the model are given by, for $|v| \geq 0.1$:

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

and for $|v| < 0.1$:

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

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

In our approach, we employed a replanning strategy to improve the adaptability of the trajectory planner. In addition to the time horizon used during the initial planning phase, 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.

### 4.2.4 Results

In this work, we compare two modeling approaches for vehicle dynamics in scenarios where road alignment plays a key role. Specifically, we evaluate a point mass model that is aligned to the road and a kinematic bicycle model that captures the vehicle's geometry and steering kinematics more explicitly.

#### Point Mass (Road-Aligned) Model

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

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

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

#### Kinematic Bicycle Model

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

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

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

#### Comparative Summary

When evaluating performance for standard driving maneuvers and moderate turning radii, the kinematic bicycle model often yields more faithful estimates of the vehicle trajectory than the point mass model, owing to its explicit handling of vehicle geometry. The point mass model, however, can be beneficial in path planning or control algorithms that rely on computationally inexpensive dynamics and do not require precise yaw-rate or steering-angle predictions.

In short, the road-aligned point mass model is a higher-level abstraction that excels in simplicity and speed but sacrifices accuracy in situations with larger lateral dynamics. The kinematic bicycle model achieves better fidelity by incorporating the essential geometry of steering, but at a slightly higher computational and implementation cost. Both models can be useful depending on the complexity and fidelity demands of the application.

## 4.3 Challenges Encountered

### 4.3.1 Replanning Challenges for Point Mass

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

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

### 4.3.2 Lag in Dynamics

### 4.3.3 Lag in Dynamics

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

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

### 4.3.4 McCormick Relaxation

To illustrate the application of these relaxations in practice, consider a path-planning scenario with $v_{min} = 1$, $v_{max} = 4$, and $v_{start} = 1$. In this scenario, the bilinear term $v\xi$, which appears in the equation of motion for $\dot{n} = v \sin \xi \approx v\xi$, is approximated using McCormick relaxations.

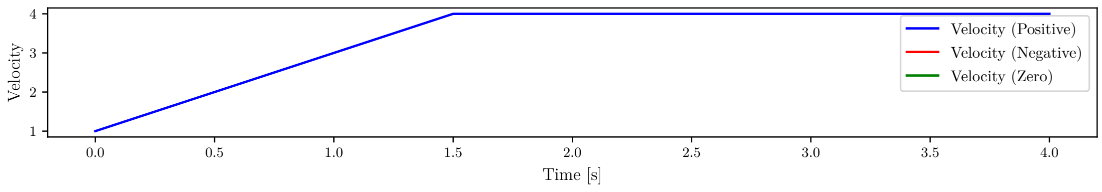


Figure 4.2: Planned velocity profile.

Figure 4.2 shows the planned velocity profile, which quickly reaches its upper limit.

Figure 4.3 depicts the alignment error $\xi$ at each planned time point. Here, $\xi$ is bounded within $-45° \leq \xi \leq 45°$. It is noteworthy that $\xi$ does not reach these bounds.

Figure 4.4 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

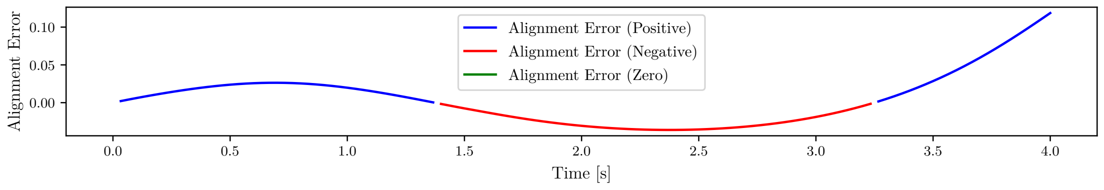


Figure 4.3: Alignment error $\xi$ over time.

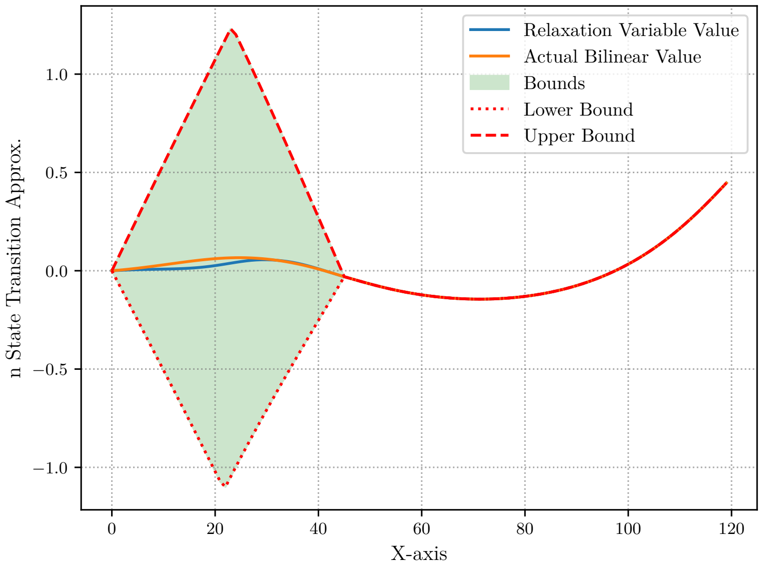


Figure 4.4: State transition approximation for $n$ using bilinear term relaxation.

state transition of $n$. Once the velocity reaches its limit, the approximation becomes increasingly accurate.

Note: The x-axis does not represent time directly but instead shows discrete time points. Here, 30 time points per second were chosen, resulting in a range from 0 to 120 for this figure, whereas the other figures range from 0 to 4.

### Main Problem with Mccormick

The usual way how I apply McCormick is by saying:

Given a State, which is valid, find the next State which is valid with the corresponding control which is also valid under a given Objective.

Problem: This is not convex.

By Convexifying this Relationship, we decouple the interaction.

Question:
Can this approach result in two states, where no valid control input can exist? If so, how often does this happen.
Lets look at a example:

$$x_{n+1} = x_n + u_n v_n$$

With State $x$ and Control Inputs $u$, $v$.
Applying Mccormick:
$w = xy$

Hesse-Matrix of $f(x, y) = xy$:

$$\nabla f = \begin{bmatrix} y \\ x \end{bmatrix}$$

$$\nabla^2 f = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$| \begin{bmatrix} -\lambda & 1 \\ 1 & -\lambda \end{bmatrix} | = \lambda^2 - 1 = 0 \implies \lambda = \pm 1$$

# 5 Discussion and Future Work

# 6 Conclusion