TECHNICAL UNIVERSITY
OF MUNICH

DEPARTMENT OF
INFORMATICS

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

DEPARTMENT OF
AERONAUTICS AND ASTRONAUTICS

Master's Thesis in Robotics, Cognition and Intelligence

# Minimizing Collision Risk in Density-based Motion Planning

Laura Lützow

TECHNICAL UNIVERSITY
OF MUNICH

DEPARTMENT OF
INFORMATICS

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

DEPARTMENT OF
AERONAUTICS AND ASTRONAUTICS

Master's Thesis in Robotics, Cognition and Intelligence

# Minimizing Collision Risk in Density-based Motion Planning

## Minimierung des Kollisionsrisikos in dichte-basierter Bewegungsplanung

| | |
|---|---|
| Author: | Laura Lützow |
| Supervisor: | Prof. Dr.-Ing. Matthias Althoff, TUM |
| Advisor: | Assistant Prof. Chuchu Fan, MIT |
| Submission Date: | September 15th, 2022 |

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

Ich versichere, dass ich diese Masterarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Munich, September 15th, 2022 Laura Lützow

# Acknowledgments

This thesis summarizes the research which I did during the last half year at the Reliable Autonomous Systems Lab at Massachusetts Institute of Technology. First, I want to thank Prof. Matthias Althoff who showed great trust in my abilities and helped me with organizing my master thesis abroad. Without him, the opportunity to write my thesis at MIT would not have arisen. Second, I want to thank my supervisor at MIT, Assistant Prof. Chuchu Fan, for giving me the chance to write my master thesis in her Lab and for the guidance and support she provided during my research.

Another big thanks goes to my lab mates, Sydney Dolan, Charles Dawson, Yue Meng, Songyuan Zhang and Kathleen Xu, who welcomed me with open arms and helped me with settling in. For my research I was supported by Yue Meng and Andres Chavez Armijos and I am very grateful for their advice during project meeting and the help they provided for transforming the thesis into a conference submission. I also want to thank everyone from VISTA, the vising student association at MIT, and SMITE, the Ultimate Frisbee team, who made my time in Boston so remarkable. In particular, I want to mention Huyen-Tram Tran, Giulia Pozzi, Jonas Oldenstädt, Tobias Lucas, Chris Quinn and David Werder.

As my studies will end with the submission of my thesis, I also want to thank everyone who supported me in the last six years. A big thanks goes to my friends and fellow students for making the last six years the most exciting time in my life. Especially, I want to thank Sarah Löcklin, Elisabeth Piontkowskie, Stefan Menger, Maximilian Kloppe, Finn Süberkrüb, Maxime Thibault, Victoria Stoisser and Julia Panzer.

My family deserves one of my greatest thanks. I want to thank my parents, Pia and Volker, and my sisters, Svenja and Lisa, for their unconditional support and for always being there for me.

# Notations

## Abbreviations

| | |
|---|---|
| **CCM** | control contraction metric |
| **CRI** | collision risk increase |
| **FPE** | Fokker-Planck equation |
| **GCI** | goal cost increase |
| **ICI** | input cost increase |
| **LE** | Liouville equation |
| **MPC** | model predictive control |
| **PDE** | partial differential equation |

## Conventions

| | |
|---|---|
| $a$ | scalar |
| $\mathbf{a}$ | vector |
| $\mathbf{a}^T$ | transposed vector $\mathbf{a}$ |
| $\|\|\mathbf{a}\|\|$ | Euclidean norm of vector $\mathbf{a}$ |
| $\mathbf{A}$ | matrix |
| $\dot{\square}$ | first time derivative of scaler, vector or matrix $\square$ |
| $\hat{\square}$ | approximated value of $\square$ |
| $\square(t)$ | $\square$ at time $t$ |
| $\square(\cdot)$ | time trajectory of $\square$ |

## Subscripts and Superscripts

| | |
|---|---|
| $\mathbf{a}_i$ | element in the $i$th row of vector $\mathbf{a}$ |
| $\mathbf{A}_{:i}$ | $i$th column of matrix $\mathbf{A}$ |
| $\mathbf{A}_{ij}$ | element in the $i$th row and $j$th column of the matrix $\mathbf{A}$ |
| $\square_*$ | reference |
| $\square^{(i)}$ | sample $i$ |

## Variables

| | |
|---|---|
| $a$ | longitudinal acceleration |
| $c_x$ | $x$-position in grid coordinates |
| $c_y$ | $y$-position in grid coordinates |
| $G_x$ | gradient of obstacle occupation probabilities in $x$-direction |
| $G_y$ | gradient of obstacle occupation probabilities in $y$-direction |
| $g$ | density concentration function |
| $g_{\mathbf{log}}$ | logarithmic density concentration function |
| $J$ | cost for trajectory optimization |
| $\mathscr{L}$ | loss for neural network training |
| $\mathbf{M}$ | contraction metric |
| $N$ | number of prediction time points |
| $P_{\mathbf{coll}}$ | collision probability between obstacles and the ego vehicle |
| $P_{\mathbf{ego}}$ | occupation probability by the ego vehicle |
| $P_{\mathbf{occ}}$ | occupation probability by obstacles |
| $p_x$ | $x$-position in real-world coordinates |
| $p_y$ | $y$-position in real-world coordinates |
| $r_{\mathbf{tube}}$ | radius of the tube in tube-based model predictive control |
| $s$ | step size |
| $S$ | number of samples |
| $t$ | time |
| $t_k$ | $k$th discrete time point |
| $\mathbf{u}$ | input vector |
| $\mathbf{U}_p$ | input parameters |
| $\mathbf{W}$ | dual contraction metric |
| $v$ | velocity |
| $\mathbf{x}$ | state vector |
| $\alpha$ | weighting factors |
| $\delta_{\mathbf{x}}$ | infinitesimal displacement between the solution and a neighboring one |
| $\Delta t$ | time increment between two consecutive time points |
| $\theta$ | heading angle |
| $\theta_{\mathbf{bias}}$ | sensor bias on heading angle measurement |
| $\lambda$ | contraction rate |
| $\rho$ | density |
| $\mathbf{\Phi}$ | flow map |
| $\omega$ | yaw rate |

# Abstract

This thesis presents a novel approach for density-based motion planning in dynamic environments. Many state-of-the-art motion planners have difficulties to reach the target in crowded, uncertain environments while keeping the collision probability small. Thus, the main objective for the proposed motion planner is to find trajectories which lead to a target position with minimal collision risk. As we additionally consider an uncertain initial state in form of a given density distribution, we will utilize density-based reachability, i.e., we will estimate the state density distribution which will be reached in the future and use it to compute and minimize the collision risk.

The proposed approach consists of three main components: First, a tracking controller will be synthesized such that trajectories starting from all possible initial states stay in the vicinity of a reference trajectory. To guarantee good tracking performance, even under disturbances, we will use contraction theory. The second component is a neural network which approximates the state density distribution for the closed-loop dynamics along the reference trajectory. Finally, a gradient-based optimization procedure will be used to optimize the reference trajectory in order to minimize the collision risk.

To evaluate the performance, the motion planning approach is applied to an autonomous car and we show that the approach outperforms state-of-the-art motion planners in a large number of dynamic environments. Furthermore, we demonstrate that the concept can be applied to real-world data without modification.

# Contents

# 1. Introduction

Every day, 3287 people on average die in car accidents making these accidents the ninth leading cause of death[1]. As more than 60% of them occur because of human errors [1], the application of autonomous cars is promised to significantly decrease this number. However, before autonomous cars can be used in everyday life, they have to meet certain safety standards. For one, we want to have guarantees that the car stays in the admissible state space, i.e., it should stay on the street and respect the speed limit. Furthermore, we want to reach the goal in a reasonable amount of time, and finally, collisions with obstacles or other traffic participants should be avoided by all means. Therefore, providing safety certificates is one key challenge in motion planning for autonomous systems.

Because of uncertainties such as measurement errors, external disturbances or model errors, we cannot precisely predict the future state of the vehicle and the environment which leads to more difficulties in making safety statements about a planned trajectory. To deal with these uncertainties, there are two main directions for motion planning: Conservative approaches focus on safety by considering all possible situations, e.g., with reachability analysis [2], by planning a safe trajectory for the worst case [3] or by computing safe sets in which safety can be guaranteed for a contained trajectory [4, 5]. Alternatively, many motion planning approaches target probabilistic safety. By predicting the most likely future states of the environment, they can plan less conservative trajectories which are safe with a high probability [6, 7]. As unsafe behavior of autonomous systems can lead to collisions and hence, could result in serious damage, only having probabilistic guarantees may not seem satisfactory in numerous applications. However, there are many cases in which a guaranteed safe trajectory cannot be found. For instance, we usually cannot find a path with zero collision probability for an autonomous car in crowded real-world environments. While in many situation we prefer to stop or revert to a fail-safe emergency maneuver [8] if the collision probability is nonzero, there are other situations where a higher collision risk is acceptable. Finding the trajectory with the smallest collision risk is in these situations of utmost importance and topic of this thesis.

The problem formulation as well as a description of the solution concept will be given in Section 1.1, while Section 1.2 presents an overview of related work. In Section 1.3, the contributions of this thesis are listed. The thesis outline is provided in Section 1.4.

[1] https://safer-america.com/car-accident-statistics/#Global, Accessed: 2022-08-29

## 1.1. Objectives

In the course of this thesis, a motion planning approach for autonomous systems under state uncertainties is developed. The exact problem will be described in Section 1.1.1, while Section 1.1.2 presents an overview of the proposed solution.

### 1.1.1. Problem Formulation

This thesis solves the problem of defining an optimal control policy for a given autonomous system and environment setup. The system shall be steered to the goal state while minimizing the collision probability with possible dynamic obstacles. Furthermore, the state and input constraints must be satisfied, and the system dynamics can be nonlinear in the states.

In contrast to standard motion planning problems, we consider an uncertain initial state following an arbitrary but known probability density function. During the execution of the control policy, the state can be measured and used for closed-loop control. However, because of limited computational power and the time constraints for real-time executability, the online computations of the controller have to be simple. Furthermore, we consider imperfect measurements due to sensor bias.

Additionally, probabilistic predictions for the evolution of the environment are given. Namely, we know for each position in the environment the probability of it being occupied by an obstacle at a certain point in time. The position of the vehicle at this time depends on the initial state and hence also follows a certain probability distribution. The overlap of this state distribution with the predicted obstacle positions yields the collision probability.

Thus, the algorithm should find a general control strategy which minimizes the cost function and satisfies the constraints for the uncertain initial states despite sensor errors. The motion planning problem is summarized in Fig. 1.1 and visualized in Fig. 1.2.

### 1.1.2. Proposed Solution

We will split our motion planning approach into three phases:

- **Preparation Phase**: This phase contains all computations and preparations which are independent of the environment setup (they only depend on the dynamics of the autonomous system) and is executed prior to the actual motion planning process. Since these computations have to be done only once for a given dynamical system, the preparation phase does not have any strict restrictions on its computational complexity.

- **Planning Phase**: In this phase, we want to find the optimal control strategy for a certain environment setup which is specified by the initial density distribution, the goal state and the environment predictions. Directly after the planning phase,

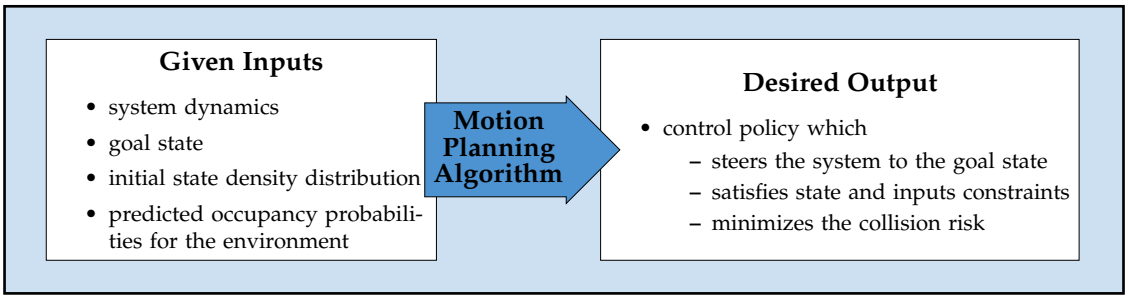
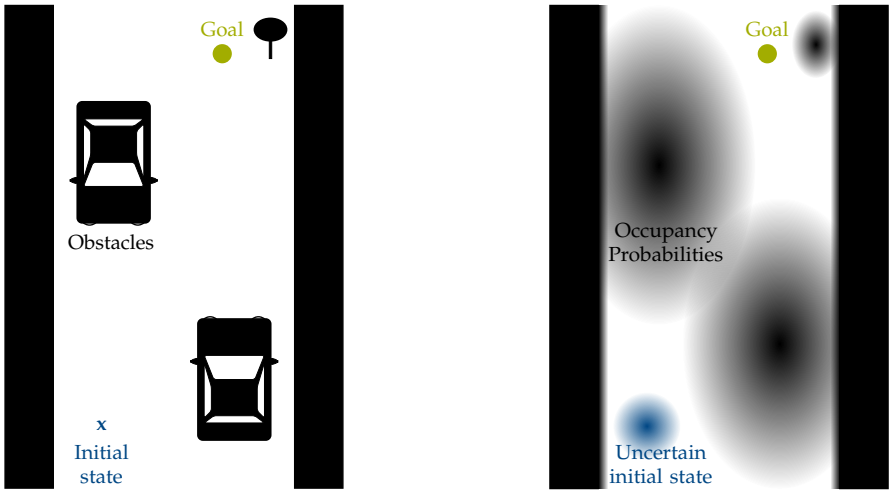



Figure 1.1.: Problem description.




(a) Standard motion planning problem: The initial state and the environment is considered as exactly known.

(b) Motion planning problem considered in this thesis: The initial state is uncertain and given by an arbitrary probability distribution. The environment is uncertain and described by occupancy probabilities.

Figure 1.2.: Visualization of the motion planning problem. We want to find a control strategy which steers the system to the goal state for all possible initial states while satisfying the constraints and minimizing the collision risk.

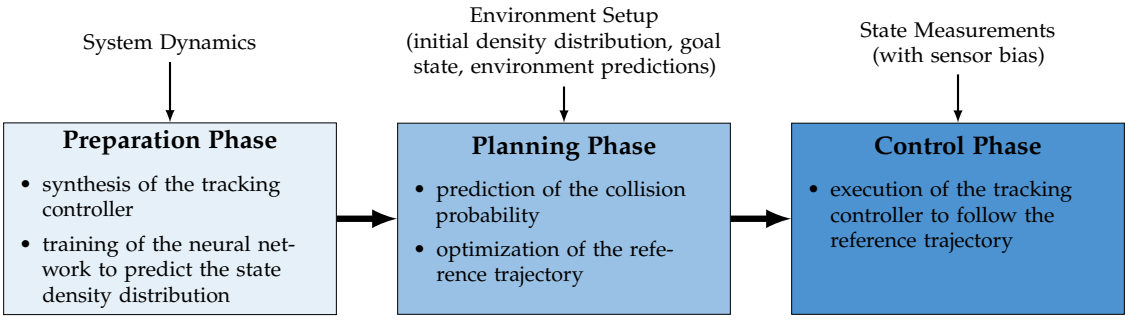



Figure 1.3.: Phases of the motion planning algorithm

the planned motion gets executed. As short planning times are desired, the computational complexity should be low.

- **Control Phase**: This phase denotes the online execution of the planned control strategy by the autonomous system. In this phase, we can only do minor computations like computing the output of a simple feedback controller.

An overview of the phases and their elements is provided in Fig. 1.3. The proposed motion planning approach can now be described as follows:

In the preparation phase, we design a tracking controller which will use real-time state measurements to track a reference trajectory. To guarantee that the considered system stays in the neighborhood of the reference trajectory despite the initial state uncertainties and disturbances, we will use contraction theory for the controller synthesis. The synthesis has to be done only once since the controller can be used for all reference trajectories and hence for all environment setups, as long as the dynamic system stays the same. By assuming the execution of the tracking controller in the control phase, the problem for the planning phase is simplified to finding the optimal reference trajectory. As we want to have short planning times, we need an efficient way to compute the collision probability along this reference trajectory. On this account, we will train a neural network which predicts the evolution of the state density distribution dependent on the reference trajectory and the initial state distribution.

In the planning phase, we optimize the reference trajectory by minimizing a differentiable collision risk with a gradient-based algorithm. The collision risk can be computed efficiently by using the density predictions of the neural network.

In the control phase, the tracking controller is applied such that the system follows the optimized reference trajectory in real-time. The controller uses state measurements which can be inaccurate because of sensor bias.

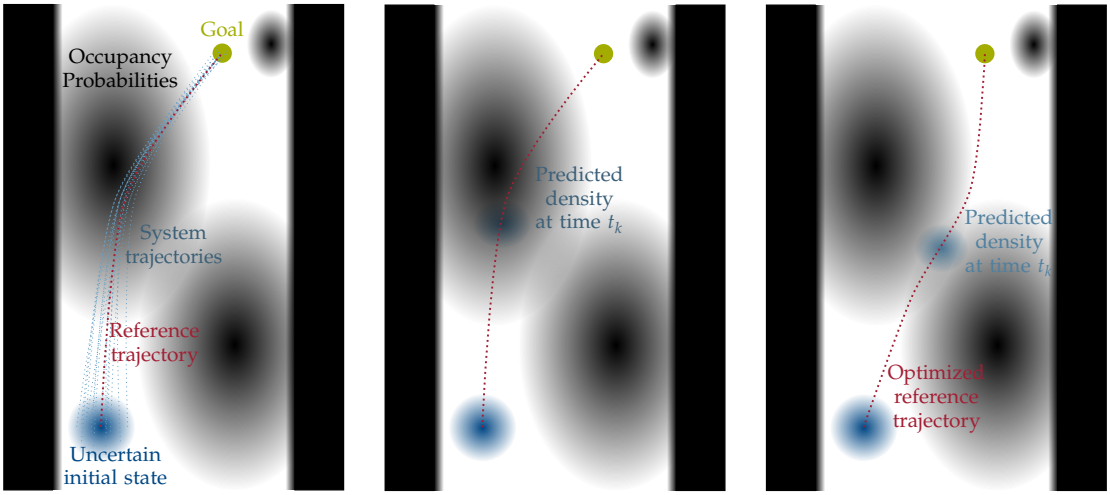
An example environment and the solution strategy is visualized in Fig. 1.4. For better visualizability, only stationary obstacles are considered.

## 1.2. Related Work

In the following, the state of the art and a review of relevant literature is provided. Firstly, the general topic of motion planning in uncertain environments is investigated in Section 1.2.1. Section 1.2.2 presents methods to predict or control the state density function for dynamical systems, followed by a short overview of related work using contraction theory in Section 1.2.3.

### 1.2.1. Motion Planning under Uncertainty

To plan safe trajectories for autonomous systems, the state and evolution of the environment has to be considered, e.g., in autonomous driving the movement of other traffic participants has to be predicted.




(a) Step 1: The control policy is defined. We will use a tracking controller such that all possible system trajectories stay close to a given reference trajectory.

(b) Step 2: Dependent on the reference trajectory, the density distribution for the controlled system can be predicted at each time step.

(c) Step 3: The reference trajectory is optimized on the basis of the density predictions in order to minimize the collision risk.

Figure 1.4.: Proposed solution. We want to find a control strategy which minimizes the collision probability for all possible initial states. The collision probability can be computed from the positional overlap between the state density distribution of the considered system (blue) and the possible positions of obstacles specified by the occupancy probabilities (black).

The authors of [9] differentiate between physic-based methods which use dynamic or kinematic models and Bayesian filters, and methods with learning components for improved modelling of long term dependencies. The output of the prediction model can be the intention of the pedestrians (e.g., crossing the street), trajectories or occupancy maps. For our motion planning approach, we consider occupancy maps which assign to each location on the map a probability that this location is occupied. These maps can be obtained from neural networks [10], Markov chains [11] or via reachability analysis [12].

The objective in motion planning is to find a trajectory which minimizes some cost function while leading the system to a specified state, satisfying the kinematic and input constraints and avoiding collision. There is a vast amount of publications dealing with this problem while considering environment predictions in the form of occupancy maps. These publications use probability navigation functions [13] which attract the system to the target position while repelling it from areas with high collision probability, chance-constrained RRT [14] or general constrained optimization techniques [15]. All of these methods assume that the initial state is exactly known, but this knowledge is often not given in the real world. Because of sensor uncertainties we can just make

probabilistic guesses about the initial state. As a consequence, it is desirable to use the probability density distribution of the initial system state for motion planning.

### 1.2.2. Density Control

There are many publications doing motion planning for Gaussian distributed initial states. Under linear dynamics, just the mean and covariance of the density distribution will change over time [16]. As it is possible to control the first two moments separately and independently, systems can be easily steered to arbitrary Gaussian distributions. The corresponding optimal control problems can be solved by convex programming [17].

As soon as nonlinear dynamics or non-Gaussian distributions are considered, controlling the density distribution becomes more challenging. Several publications deal with the covariance steering problem for nonlinear systems where they try to find the input signal to change the covariance of a Gaussian state distribution in a predefined way [18, 19, 20]. However, these methods are not directly applicable to non-Gaussian state distributions.

The control of arbitrary density distributions can be closely related to optimal transport theory where a given distribution is rearranged while the cost of transport is minimized [21]. In [22], the optimal state feedback policy for this task is obtained by solving the Hamilton-Jacobi-Bellman equation. This is achieved by reformulating the control problem as a Schrödinger bridge problem and solving the Schrödinger system with fixed point recursion. Additionally, state constraints as required for obstacle avoidance can be considered [23]. By capitalizing the differential flatness of a kinematic bicycle model, this density control method can also be applied to safe lane changing in autonomous driving [24]. In addition to the fact that this approach can only be used for full state feedback linearizable systems, the convergence of the fixed-point recursion cannot be guaranteed.

Instead of optimizing the density and motion trajectory at once, an iterative optimization strategy can be utilized by splitting the problem into the density prediction part and the subsequent policy optimization:

In [25], a duality relationship between the density function which follows the Liouville equation and the value function of optimal control problems is proved. Thus, the problem can be solved with a primal dual algorithm alternating between the solution of the Liouville equation to get the density for a given control policy and updating the policy with the Hamilton-Jacobi-Bellman equation. However, the computational effort for solving the Hamilton-Jacobi-Bellman equation is quite high.

The duality of the density function to Q functions is shown and leveraged in [26]. By optimizing a modified Q function with model-free reinforcement learning, time-invariant density constraints are enforced and the convergence to a near-optimal solution can be proved.

In [27], the complexity of solving the Hamilton-Jacobi-Bellman equation is circumvented by using a perturbation method and nonlinear programming. A reference

trajectory is generated and tracked by a tracking controller. The density distribution along this reference trajectory is predicted and if the resulting collision probability is smaller than a certain threshold, the trajectory is considered safe and gets accepted. Otherwise, the trajectory segment where a high collision probability was detected is perturbed and then checked again. One big limitation of this approach is its computation time - the collision checking is computationally complex and it cannot be guaranteed that a safe trajectory in the neighborhood of the initial trajectory will be found nor how many iterations will be needed. Furthermore, this approach only tries to reach a certain safety level, but no optimality statements can be made. This thesis uses a similar approach to [27] with more efficient collision probability computations and an improved optimization algorithm. Additionally, a tracking controller with convergence guarantees will be synthesized by using contraction theory.

### 1.2.3. Contraction Theory

The foundations for contraction theory were laid by Winfried Lohmiller and Jean-Jacques E. Slotine [28] providing a strong tool for analyzing the stability of nonlinear systems and synthesizing guaranteed stable controllers. While contraction controllers were extensively applied to dynamic systems with stochastic disturbances [29, 30, 31], only few publications consider uncertain initial conditions. In [32], the initial density distribution is over-approximated by a ball induced by the Finsler distance. Using contraction theory, the contraction rate of this ball can be computed such that the maximum extensions of the density distribution can be calculated for each point in time.

How the density distribution of linear systems can be steered to a prescribed distribution shape is analyzed in [33]. However, this method cannot be applied to general nonlinear systems.

To the best of the authors knowledge, there is no work combining contraction theory with density control for nonlinear systems.

## 1.3. Contributions

Our motion planning approach has several advantages over prior work: The approach is valid for arbitrary initial probability distributions and nonlinear system dynamics. By predicting the probability distribution at each point in time, the approach is not over-cautious and can provide good estimates for the collision probabilities. Because of the neural network approximation of the closed-loop dynamics and the density function, and an efficient optimization framework, small planning times can be achieved. The resulting control strategy can be executed in real-time with low computational complexity. Additionally, the resulting trajectory is optimal in a given cost criteria and the gradient-based optimization method can easily overcome bad local minima by comparing the results of different start conditions.

Other main contributions of this thesis are:

1. We are the first to connect contraction theory with density control for nonlinear systems.
2. We provide an efficient method to compute the collision probability for closed-loop dynamics and an effective gradient-based algorithm for its optimization.
3. We apply the approach to a simulated autonomous vehicle and compare its performance with state-of-the-art motion planning methods in a large number of time-variant environments.
4. We validate the approach with real-world data.

## 1.4. Outline of the Thesis

After having introduced the topic and related work in Chapter 1, Chapter 2 provides an overview of the theoretical fundamentals used in this thesis. The basics for predicting the state density distribution are explained. In this regard, we present the Fokker-Plank equation and the Liouville equation and describe solution methods. In the second part of the chapter, an introduction to contraction analysis is given and important results for synthesizing a contraction controller are derived.

Chapter 3 deals with the computation of the collision probability. First, we introduce the neural contraction controller which is used to track the reference trajectory and describe the training process. Next, the neural network to predict the state density distribution along the reference trajectory is presented, followed by the description of the collision probability estimation procedure.

The optimization method to minimize the collision probability is discussed in Chapter 4. The composition of a differentiable cost function is explained and the cost computation algorithm is given. Furthermore, we describe the gradient-based optimization method which consists of an initialization process to overcome local minima and a subsequent optimization algorithm to minimize the overall collision probability.

In Chapter 5, the motion planning approach is applied to an autonomous car and time-variant environments. First, the implementation is described and the performance of the controller, of the density predictor and of the optimization method is analyzed. Additionally, an ablation study for the optimization method is given where the gradient-based approach is compared with a search-based and a sampling-based method.

Finally, we evaluate the performance of the overall motion planning approach in Chapter 6. We explain the necessity of including the collision probability in the cost function and describe state-of-the-art motion planners which can be used as baselines. Simulation results are provided for artificially generated environments and for environments based on real-world data.

Chapter 7 concludes the thesis. Limitations of the motion planning approach are given and an overview of possible future research directions is presented.

# 2. Preliminaries

To plan safe trajectories for autonomous systems, we have to consider the uncertainties of the real world. These uncertainties can result from imperfect measurements or model inaccuracies since generally the environment and the system can never be modeled exactly. This thesis concentrates on uncertainties in the initial state in form of a given probability density distribution. To predict the evolution of the probability density function, the Liouville equation is solved with a neural network. To track the reference trajectory despite the initial state uncertainties a contraction controller is leveraged. The fundamentals of these topics are presented in this chapter:

The basics about the evolution of the density function are given in Section 2.1 where the first part deals with the general Fokker-Planck equation and the second part focuses on the solution of the Liouville equation to approximate the density for deterministic systems. The chapter concludes with a brief introduction to contraction theory in Section 2.2.

## 2.1. Density Evolution

The density function describes the distribution of states in the state space. If the integral of the density function over the whole state space sums up to one, this function can be called probability density function. Thus, its value $\rho(\mathbf{x}, t)$ denotes the probability density that the system resides around state $\mathbf{x}$ at time $t$. To predict the collision probability of the system for a given control input, the evolution of the density function must be computed.

In general, the evolution of the density function follows the Fokker-Planck equation which is the topic of Section 2.1.1. Under certain assumptions, this equation can be simplified to the Liouville equation which is discussed in Section 2.1.2.

### 2.1.1. Fokker-Planck Equation

The Fokker-Planck equation (FPE) describes the evolution of the density function for dynamic systems in the presence of uncertainties. In this subsection, we consider the general dynamical system which follows the stochastic differential equation

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) + \mathbf{G}(\mathbf{x}, \mathbf{u})\mathbf{\Gamma}(t), \tag{2.1}$$

where $\mathbf{x}(t)$ is the state, $\mathbf{u}(t)$ is the control input, $\mathbf{\Gamma}(t)$ is a Gaussian white noise process with the correlation function $\mathbf{Q}\delta(t_1 - t_2)$ and $\delta(\cdot)$ is the Dirac delta function. The

arguments of $\mathbf{x}$ and $\mathbf{u}$ are omitted in this chapter due to better readability. The density distribution $\rho(\mathbf{x}, t)$ of this stochastic system satisfies the FPE [34, 35]:

$$\frac{\partial}{\partial t}\rho(\mathbf{x},t) = -\sum_{i=1}^{n} \frac{\partial}{\partial \mathbf{x}_i}\Big(\mathbf{f}(\mathbf{x},\mathbf{u}) + \frac{1}{2}\frac{\partial \mathbf{G}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}\mathbf{Q}\mathbf{G}(\mathbf{x},\mathbf{u})\Big)_i \, \rho(\mathbf{x},t) \tag{2.2}$$

$$+\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2}{\partial \mathbf{x}_i \partial \mathbf{x}_j}\big(\frac{1}{2}\mathbf{G}(\mathbf{x},\mathbf{u})\mathbf{Q}\mathbf{G}^T(\mathbf{x},\mathbf{u})\big)_{ij} \, \rho(\mathbf{x},t), \tag{2.3}$$

where $\square_{ij}$ is the element in the $i$th row and $j$th column of the matrix $\square$ and $\square_i$ denotes the element in the $i$th row of the vector $\square$. Since the solution should be a valid probability distribution, it has to be positive everywhere and its integral over the whole state space should be one.

Even by assuming a constant diffusion coefficient, i.e., $\mathbf{G}(\mathbf{x},\mathbf{u}) = \mathbf{G}$, which corresponds to additive Gaussian noise, this second-order partial differential equation (PDE) is very difficult to solve. The analytical solution can only be obtained for special dynamical models under strict conditions. Numerical methods for approximating the solution such as finite element methods or finite difference methods are prone to discretization error and often require substantial computing resources, especially when computing in two- or higher-dimensional domains [35]. As an alternative, the FPE can be solved with deep learning by directly enforcing the PDE and the positivity and normalization constraints on the outputs of an artificial neural network. However, this was previously done only for the stationary FPE, i.e., $\frac{\partial}{\partial t}\rho(\mathbf{x},t) = 0$, [36, 35, 37] or for systems with less than three dimensions [38]. Furthermore, no work considered the dependence of the system dynamics on some control input. Consequently, for exactly approximating the density evolution of a controlled stochastic system with more than two dimensions, a very big neural network and large training times would be necessary. Thus, we will assume in the following that the stochastic noise in Eq. (2.1) is negligible resulting in a much simpler evolution equation.

### 2.1.2. Liouville Equation

For a general deterministic system

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u}), \tag{2.4}$$

the FPE simplifies to the Lioville equation (LE)

$$\frac{\partial \rho(\mathbf{x},t)}{\partial t} = -\sum_{i=1}^{n} \frac{\partial}{\partial \mathbf{x}_i}\mathbf{f}_i(\mathbf{x},\mathbf{u}) \, \rho(\mathbf{x},t), \tag{2.5}$$

a quasi-linear, first-order PDE [39]. While the closed-form solution of the LE cannot be calculated for most systems, the density can be easily evaluated along trajectories by transforming the PDE to the ordinary differential equation [25]

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\rho}(\mathbf{x},t) \end{bmatrix} = \begin{bmatrix} f(\mathbf{x},\mathbf{u}) \\ -(\nabla_{\mathbf{x}} \cdot f(\mathbf{x},\mathbf{u}))\rho(\mathbf{x},t) \end{bmatrix}, \tag{2.6}$$

where $\nabla_{\mathbf{x}} \cdot \big(\rho(\mathbf{x},t) \cdot f(\mathbf{x},\mathbf{u})\big) = \sum_{i=1}^{n} \frac{\partial}{\partial \mathbf{x}_i} \mathbf{f}_i(\mathbf{x},\mathbf{u})\, \rho(\mathbf{x},t)$ is the divergence of the vector field $\rho(\mathbf{x},t) \cdot f(\mathbf{x},\mathbf{u})$. Thus, for a given initial density distribution $\rho_0$, the density evolution can be predicted by uniformly sampling initial conditions $\mathbf{x}_0$ from the support of $\rho_0$ and solving Eq. (2.6) at the prediction time $t_p$ with initial condition $\mathbf{x}(0) = \mathbf{x}_0$ and $\rho(\mathbf{x},0) = \rho_0(\mathbf{x}_0)$. With interpolation methods the whole density distribution at time $t_p$ can be constructed [40].

In [41], the prediction process is accelerated by training a neural network to solve Eq. (2.6). Since the closed-form solution for the density

$$\rho\big(\mathbf{\Phi}(\mathbf{x}_0,\mathbf{u},t),t\big) = \rho_0(\mathbf{x}_0) \underbrace{\exp\Big(-\int_0^t \nabla_{\mathbf{x}} \cdot \mathbf{f}\big(\mathbf{\Phi}(\mathbf{x}_0,\mathbf{u},\tau),\mathbf{u}\big) d\tau\Big)}_{g(\mathbf{x}_0,\ \mathbf{u},\ t)} \tag{2.7}$$

is linear in the initial condition $\rho_0(\mathbf{x}_0)$, it is sufficient to learn the flow map $\mathbf{x}(t) = \mathbf{\Phi}(\mathbf{x}_0,\mathbf{u},t)$ and the density concentration function $g(\mathbf{x}_0,\mathbf{u},t)$ from the inputs $\mathbf{x}_0$, $\mathbf{u}$ and $t$. By scaling the density concentration function of a trajectory with its initial density, the true density evolution can be recovered.

## 2.2. Contraction Analysis

To make the autonomous system track a reference trajectory, a contraction controller will be used in this thesis. This section concentrates on the theoretical fundamentals for the controller synthesis and thereby mostly summarizes the results from [28] and [42] about contraction analysis:

Contraction analysis is used for studying the stability of nonlinear systems by evaluating its differential dynamics along solutions. Convergence of all neighboring solutions to each other implies global exponential convergence to a single trajectory.

In this thesis, we consider control-affine nonlinear systems of the form

$$\dot{\mathbf{x}} = \underbrace{\mathbf{a}(\mathbf{x}) + \mathbf{B}(\mathbf{x})\mathbf{u}}_{\mathbf{f}(\mathbf{x},\mathbf{u})}. \tag{2.8}$$

It is assumed that $\mathbf{a}(\mathbf{x})$ and $\mathbf{B}(\mathbf{x})$ are smooth functions and $\mathbf{u}(t)$ is at least piecewise-continuous. A valid solution $\big(\mathbf{x}(t),\ \mathbf{u}(t)\big)$ satisfies the dynamics in Eq. (2.8) for all $t \in \mathbf{R}^+$. The reference trajectory $\big(\mathbf{x}_*(t),\ \mathbf{u}_*(t)\big)$ as the trajectory which we want our system to follow has to be a solution of Eq. (2.8).

To make statements about the behavior of trajectories with respect to each other, the differential dynamics of the system have to be analyzed. The differential dynamics are defined along the solution $\big(\mathbf{x}(t),\ \mathbf{u}(t)\big)$ according to

$$\dot{\delta}_{\mathbf{x}}(t) = \mathbf{A}(\mathbf{x},\mathbf{u},t)\delta_{\mathbf{x}}(t) + \mathbf{B}(\mathbf{x},t)\delta_{\mathbf{u}}(t), \tag{2.9}$$

where $\mathbf{A}(\mathbf{x},\mathbf{u},t) = \frac{\partial \mathbf{a}}{\partial \mathbf{x}} + \sum_{i=1}^{m} \frac{\partial \mathbf{B}_{:i}}{\partial \mathbf{x}} \mathbf{u}_i$ , $\mathbf{B}_{:i}(\mathbf{x})$ is the $i$th column of $\mathbf{B}(\mathbf{x})$ and $\delta_{\mathbf{x}}(t)$ is the infinitesimal displacement between the solution and a neighboring one at time $t$. Applying the smooth feedback control law $\mathbf{u}(t) = \mathbf{k}(\mathbf{x},t) + \mathbf{v}(t)$ and denoting the derivative

$\frac{\partial \mathbf{k}}{\partial \mathbf{x}}$ by $\mathbf{K}(\mathbf{x}, t)$ leads to the closed-loop system

$$\dot{\delta}_{\mathbf{x}}(t) = \big(\mathbf{A}(\mathbf{x}, \mathbf{u}, t) + \mathbf{B}(\mathbf{x}, t)\mathbf{K}(\mathbf{x}, t)\big)\delta_{\mathbf{x}}(t). \tag{2.10}$$

For clarity, the arguments of the functions are omitted in the following.

The rate of change of the squared displacement can be computed with

$$\begin{aligned} \frac{d}{dt}(\delta_{\mathbf{x}}^T \delta_{\mathbf{x}}) &= 2\delta_{\mathbf{x}}^T \dot{\delta}_{\mathbf{x}} && (2.11)\\ &= 2\delta_{\mathbf{x}}^T (\mathbf{A} + \mathbf{B}\mathbf{K})\delta_{\mathbf{x}} && (2.12)\\ &\leq 2\lambda_{\max} \delta_{\mathbf{x}}^T \delta_{\mathbf{x}}, && (2.13) \end{aligned}$$

with $\lambda_{\max}$ being the largest eigenvalue of $0.5\big((\mathbf{A} + \mathbf{B}\mathbf{K}) + (\mathbf{A} + \mathbf{B}\mathbf{K})^T\big)$. If $\lambda_{\max}(\mathbf{x}, t)$ is uniformly strictly negative in a region $\mathbf{x} \in \chi$ which is equivalent to $\mathbf{A} + \mathbf{B}\mathbf{K}$ being uniformly negative definite, any infinitesimal displacement in this region converges exponentially towards zero. Thus, the system is called contracting with rate $\lambda_{\max}$.

This result can be generalized to arbitrary metrics using the differential coordinate transformation $\mathbf{\Theta}(\mathbf{x}, t)$. The displacement in generalized coordinates $\delta_{\mathbf{z}}$ can be computed with

$$\delta_{\mathbf{z}} = \mathbf{\Theta}\delta_{\mathbf{x}}. \tag{2.14}$$

The time derivative of $\delta_{\mathbf{z}}$ is now defined as

$$\begin{aligned} \dot{\delta}_{\mathbf{z}} &= \dot{\mathbf{\Theta}}\delta_{\mathbf{x}} + \mathbf{\Theta}\dot{\delta}_{\mathbf{x}} && (2.15)\\ &= \big(\dot{\mathbf{\Theta}} + \mathbf{\Theta}(\mathbf{A} + \mathbf{B}\mathbf{K})\big)\mathbf{\Theta}^{-1}\delta_{\mathbf{z}} && (2.16) \end{aligned}$$

which leads to the rate of change of the squared displacement

$$\begin{aligned} \frac{d}{dt}(\delta_{\mathbf{z}}^T \delta_{\mathbf{z}}) &= 2\delta_{\mathbf{z}}^T \dot{\delta}_{\mathbf{z}} && (2.17)\\ &= 2\delta_{\mathbf{z}}^T \big(\dot{\mathbf{\Theta}} + \mathbf{\Theta}(\mathbf{A} + \mathbf{B}\mathbf{K})\big)\mathbf{\Theta}^{-1}\delta_{\mathbf{z}} && (2.18)\\ &= 2\delta_{\mathbf{x}}^T \mathbf{\Theta}^T \big(\dot{\mathbf{\Theta}} + \mathbf{\Theta}(\mathbf{A} + \mathbf{B}\mathbf{K})\big)\delta_{\mathbf{x}} && (2.19)\\ &= \delta_{\mathbf{x}}^T \big(\dot{\mathbf{M}} + \mathbf{M}(\mathbf{A} + \mathbf{B}\mathbf{K}) + (\mathbf{A} + \mathbf{B}\mathbf{K})^T \mathbf{M}\big)\delta_{\mathbf{x}}, && (2.20) \end{aligned}$$

where the last line is obtained by replacing $\mathbf{\Theta}^T\mathbf{\Theta}(\mathbf{A} + \mathbf{B}\mathbf{K})$ with its symmetric part and by setting $\mathbf{M} = \mathbf{\Theta}^T\mathbf{\Theta}$. If the controller makes the closed-loop system strictly contracting with rate $\lambda$ in the metric $\mathbf{M}$, i.e. $\frac{d}{dt}(\delta_{\mathbf{x}}^T \mathbf{M} \delta_{\mathbf{x}}) \leq -\lambda \delta_{\mathbf{x}}^T \mathbf{M} \delta_{\mathbf{x}}$, it was proven in [42] that

$$\dot{\mathbf{M}} + \mathbf{M}(\mathbf{A} + \mathbf{B}\mathbf{K}) + (\mathbf{A} + \mathbf{B}\mathbf{K})^T \mathbf{M} < -2\lambda \mathbf{M}. \tag{2.21}$$

Consequently, for $\delta_{\mathbf{x}} \neq 0$ and $\delta_{\mathbf{x}}^T \mathbf{M}\mathbf{B} = 0$, it holds that

$$\delta_{\mathbf{x}}^T \big(\dot{\mathbf{M}} + \mathbf{M}\mathbf{A} + \mathbf{A}^T\mathbf{M} + 2\lambda \mathbf{M}\big)\delta_{\mathbf{x}} < 0. \tag{2.22}$$

Thus, every displacement $\delta_{\mathbf{x}}$ which is orthogonal to the span of actuated directions $\mathbf{B}_{:i}(\mathbf{x})$ has to be naturally contracting with rate $\lambda$. An uniformly bounded metric $\mathbf{M}$ fulfilling these equations is called control contraction metric. Eq. (2.22) can be convexified with the change of variables $\boldsymbol{\eta} = \mathbf{M}\delta_{\mathbf{x}}$ and $\mathbf{W} = \mathbf{M}^{-1}$ where the matrix $\mathbf{W}$ is called the dual contraction metric. This leads to the condition

$$\boldsymbol{\eta}^T\mathbf{B} = 0 \implies \boldsymbol{\eta}^T(-\dot{\mathbf{W}} + \mathbf{A}\mathbf{W} + \mathbf{W}\mathbf{A}^T + 2\lambda\mathbf{W})\boldsymbol{\eta} < 0. \tag{2.23}$$

Furthermore, [43] shows that the tracking error between trajectory $\mathbf{x}(t)$ and reference trajectory $\mathbf{x}_*(t)$ can be upper-bounded by

$$||\mathbf{x}(t) - \mathbf{x}_*(t)||_2 \leq \sqrt{\frac{\overline{m}}{\underline{m}}} \exp(-\lambda t)\ ||\mathbf{x}(0) - \mathbf{x}_*(0)||_2 \tag{2.24}$$

if Eq. (2.21) is fulfilled and if $\mathbf{M}$ is bounded by $\underline{m}\mathbf{I} \leq \mathbf{M} \leq \overline{m}\mathbf{I}$.

# 3. Prediction of the Collision Probability

In this thesis, we want to find the control policy which minimizes the collision probability. Therefore, this chapter focuses on the steps to predict the collision probability for a given control policy, whereas the next chapter will deal with its minimization.

The control policy depends on the initial state which is not known exactly during the planning phase. Though, we want to have small optimization times and thus, we cannot plan a separate control policy for all possible initial states. Hence, we have to make restrictions about the space of possible control policies. Analog to [27], we will confine the planning algorithm to the optimization of a reference trajectory which is independent of the exact initial state but will be tracked with an online feedback controller. The synthesis of the controller is described in Section 3.1, whereas Section 3.2 focuses on the prediction of the density distribution along a given reference trajectory. Based on these density predictions, the collision probability can be computed which is the topic of Section 3.3.

## 3.1. Controller Synthesis

First, we have to create a controller to track the reference trajectory. The system dynamics for an arbitrary control-affine system are given by Eq. (2.8). A valid reference trajectory must be a solution of these dynamics, i.e., the reference trajectory $\mathbf{x}_*(t)$ satisfies

$$\dot{\mathbf{x}}_* = \mathbf{a}(\mathbf{x}_*) + \mathbf{B}(\mathbf{x}_*)\mathbf{u}_*. \tag{3.1}$$

In the following, we want to find a controller $\mathbf{u}(\mathbf{x}, \mathbf{x}_*, \mathbf{u}_*)$ which steers the trajectories of all possible initial states to the neighborhood of the reference trajectory. Since the true initial state and thus the initial deviation from the reference trajectory is unknown, the controller has to be able to deal with big tracking errors. Moreover, we would like to have formal guarantees on the convergence to ensure good tracking performance for all initial states and despite imperfect measurements, e.g., due to sensor bias.

These requirements can be met by contraction theory: If a valid control contraction metric (CCM) exists, a controller which is contracting for any reference trajectory can be computed [42]. Furthermore, bounds on the evolution of the tracking error can be provided [43].

However, the synthesis of a CCM is not a trivial task and relies on several assumptions on the structure of the dynamical system. The subsequent controller synthesis, usually conducted through the computation of the CCM's geodesics [44], is computationally challenging as well. As we would like to have a motion planning approach which is

easily applicable to different dynamic systems, we will use the method from [43] to automatically learn the contraction metric and the contraction controller:

To enforce the symmetry and the positive definiteness, the dual contraction metric $\mathbf{W}(\mathbf{x})$ is computed with

$$\mathbf{W}(\mathbf{x}) = \mathbf{N}_1(\mathbf{x})^T \mathbf{N}_1(\mathbf{x}) + \underline{w}\mathbf{I},$$

where $\underline{w}$ is the predefined lower bound for the smallest eigenvalue of $\mathbf{W}$. The matrix $\mathbf{N}_1(\mathbf{x})$ is approximated by a neural network with the input $\mathbf{x}$. Simultaneously, the contraction controller is learned using two separate neural networks $\mathbf{N}_2(\mathbf{x}, \mathbf{x}_*)$ and $\mathbf{N}_3(\mathbf{x}, \mathbf{x}_*)$ which take the sample points $(\mathbf{x}, \mathbf{x}_*)$ as input. The controller output can then be evaluated using

$$\mathbf{u}(\mathbf{x}, \mathbf{x}_*, \mathbf{u}_*) = \mathbf{N}_2(\mathbf{x}, \mathbf{x}_*) \cdot \tanh\big(\mathbf{N}_3(\mathbf{x}, \mathbf{x}_*) \cdot (\mathbf{x} - \mathbf{x}_*)\big) + \mathbf{u}_*. \tag{3.2}$$

During the training, the loss function

$$\mathscr{L}_w = C_u + C_w + C_1 + C_2$$

gets minimized. The loss term $C_u$ enforces Eq. (2.21) which describes the sufficient condition for the closed-loop system to be contracting, while the term $C_w$ tries to minimize the overshoot of the tracking error (see Eq. (2.24)). $C_1$ and $C_2$ are two auxiliary loss terms which enforce sufficient conditions for the learned metric to be a valid CCM. The latter are supposed to improve the training and guide the optimization. A more detailed description of the training process and theoretical convergence guarantees are given in [43].

To get the control signal, we only need to feed the given reference trajectory and the measured state into the trained controller network. Hence, just a small number of simple computations are necessary in the control phase.

## 3.2. Density Estimation with Neural Networks

The next step is to predict the density for the system

$$\dot{\mathbf{x}} = \mathbf{f}\big(\mathbf{x}, \mathbf{u}(\mathbf{x}, \mathbf{x}_*, \mathbf{u}_*)\big) \tag{3.3}$$

controlled by the contraction controller from Section 3.1 and dependent on the reference trajectory $(\mathbf{x}_*,\ \mathbf{u}_*)$. For simplicity, we do not consider the physical dimensions of the system in this thesis but assume that we plan for a point.

The density at time $t_k \in \{t_0, t_1, ..., t_N\}$, where $t_N$ is the prediction horizon, can be computed using the Liouville equation (see Eq. (2.5)). Since the Liouville equation usually cannot be solved analytically, the approach of [41] is utilized and a neural network is trained to approximate the density concentration function and the flow map.

The state trajectory and the evolution of its density can be computed by numerically integrating Eq. (2.6) with the forward Euler method. Since the density values get very

large very quickly, we will compute the logarithm of the density to avoid numerical issues. This leads to the Euler scheme:

$$\log(\rho(t_k)) = \log\Big(\rho(t_{k-1}) - \big(\nabla_{\mathbf{x}} \cdot \mathbf{f}(t_{k-1})\ \rho(t_{k-1})\ \Delta t\big)\Big) \tag{3.4}$$

$$= \log(\rho(t_{k-1})) + \log\Big(1 - \nabla_{\mathbf{x}} \cdot \mathbf{f}(t_{k-1})\ \Delta t\Big), \tag{3.5}$$

where $\rho(t_k)$ and $\mathbf{f}(t_k)$ denote the estimates $\rho(\mathbf{x}(t_k), t_k)$ and $\mathbf{f}\big(\mathbf{x}(t_k), \mathbf{u}(\mathbf{x}(t_k), \mathbf{x}_*(t_k), \mathbf{u}_*(t_k))\big)$ for the time $t_k$, respectively, and $\Delta t = t_{k+1} - t_k$ is the time increment.

From Eq. (2.7), we know that the change of density along a trajectory is independent of its initial density value. Consequently, it is desirable to directly estimate this change, the so-called density concentration function. Here, we concentrate on the logarithm of the density concentration function which can be described by

$$g_{\log}(\mathbf{x}(t_k), t_k) = \log\big(g(\mathbf{x}(t_k), t_k)\big) \tag{3.6}$$

$$= -\int_0^{t_k} \nabla_{\mathbf{x}} \cdot \mathbf{f}\big(\mathbf{x}(\tau), \mathbf{u}(\mathbf{x}(\tau), \mathbf{x}_*(\tau), \mathbf{u}_*(\tau))\big) d\tau. \tag{3.7}$$

On this basis, we can train a neural network to predict the density and the state evolution. Since the density and state predictions depend on the reference trajectory $\{\mathbf{x}_*(\cdot), \mathbf{u}_*(\cdot)\}$, we need an efficient representation for it that we can use as input for the neural network: Here, we parameterize the reference input trajectory with the parameters $\mathbf{U}_p$ such that $\mathbf{u}_*(\cdot)$ can be unambiguously reconstructed by

$$\mathbf{u}_*(t) = \mathbf{h}(\mathbf{U}_p, t). \tag{3.8}$$

Possible choices are polynomial parametrizations (i.e., $\mathbf{U}_p$ represents the coefficients of a predefined polynomial structure), sine and cosine representations or discretizations (i.e., $\mathbf{U}_p$ specifies the input values for certain points in time which are held constant until the next time point is reached). As the reference state trajectory $\mathbf{x}_*(\cdot)$ is fully described by the initial reference state $\mathbf{x}_*(0)$ and the input trajectory $\mathbf{u}_*(\cdot)$, we can use $\{\mathbf{x}_*(0), \mathbf{U}_p\}$ as input for the neural network.

The training data for the neural network can be generated by computing the state trajectories and the logarithmic density concentration function for various initial conditions and input parameters. Instead of the actual state $\mathbf{x}(t_k)$, we will predict the deviation of the reference trajectory $\mathbf{x}_e(t_k) = \mathbf{x}(t_k) - \mathbf{x}_*(t_k)$. This adaption changes the computational complexity only slightly and leads to better training results.

The neural network is then trained by minimizing the cost

$$\mathscr{L}(\mathbf{x}(0), \mathbf{U}_p, t_k) = \underbrace{\big(\hat{\mathbf{x}}_e(t_k) - \mathbf{x}_e(t_k)\big)^2}_{\mathscr{L}_{\mathbf{x}}} + \underbrace{\alpha_g \big(\hat{g}_{\log}(\mathbf{x}(t_k), t_k) - g_{\log}(\mathbf{x}(t_k), t_k)\big)^2}_{\mathscr{L}_g} \tag{3.9}$$

for randomly sampled initial conditions $\mathbf{x}(0)$, input parameters $\mathbf{U}_p$ and points in time $t_k$, where $\hat{\mathbf{x}}_e(t_k)$ and $\hat{g}_{\log}(\mathbf{x}(t_k), t_k)$ are the neural network predictions for the state deviation

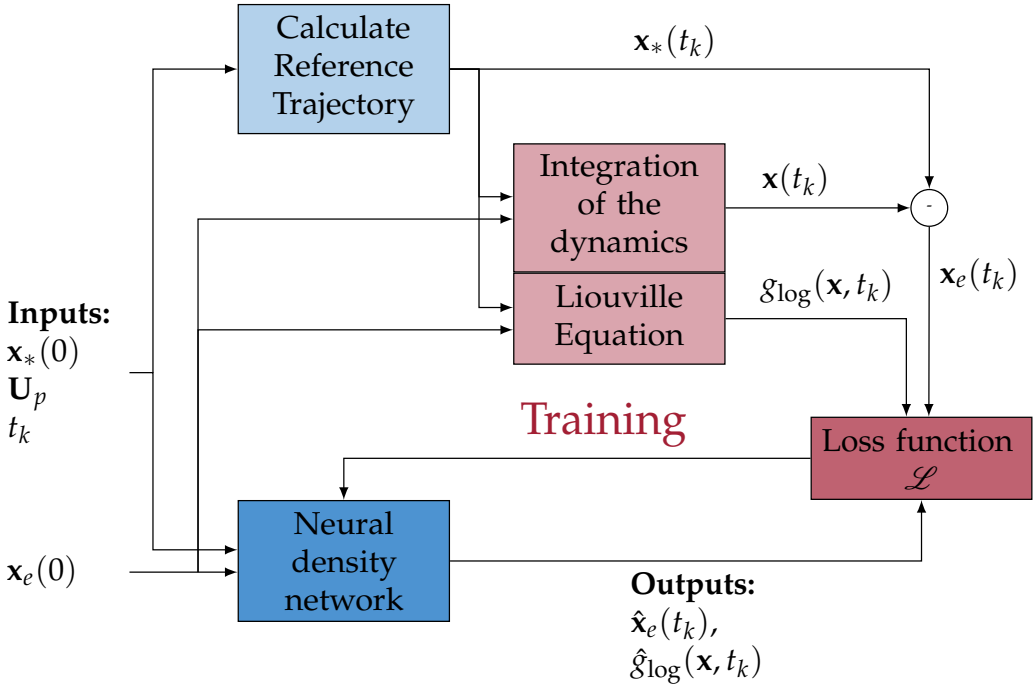



Figure 3.1.: Training of the density neural network. We sample initial reference states $\mathbf{x}_*(0)$, input parameters $\mathbf{U}_p$, prediction time points $t_k$ and initial deviations of the reference trajectory $\mathbf{x}_e(0)$ and use them as input for the neural network. The network outputs the deviation of the reference trajectory at time point $t_k$, $\hat{\mathbf{x}}_e(t_k)$, and the logarithmic density function at time $t_k$, $\hat{g}_{\log}(\mathbf{x}, t_k)$. The true values for $\mathbf{x}_e(t_k)$ and $g_{\log}(\mathbf{x}, t_k)$ can be estimated by integrating the system dynamics and solving the Liouville equation, respectively. Finally, the network weights can be optimized by minimizing the loss function from Eq. (3.9).

and the logarithmic density concentration function, respectively. The training process and the neural network inputs and outputs are visualized in Fig. 3.1.

By knowing the initial density $\rho(\mathbf{x}(0), 0)$ at state $\mathbf{x}(0)$, the density $\hat{\rho}(\mathbf{x}(t_k), t_k)$ can be predicted with Eq. (2.7) which leads to

$$\hat{\rho}(\mathbf{x}(t_k), t_k) = \rho(\mathbf{x}(0), 0) \ \exp\left(\hat{g}_{\log}(\mathbf{x}(t_k), t_k)\right) . \tag{3.10}$$

Finally, the overall density distribution $\rho(\cdot, t_k)$ can be approximated by sampling a large number of initial conditions from the support of the initial density distribution, predicting their density evolution with the trained neural network and interpolating and normalizing the results. This procedure is illustrated in Fig. 3.2.

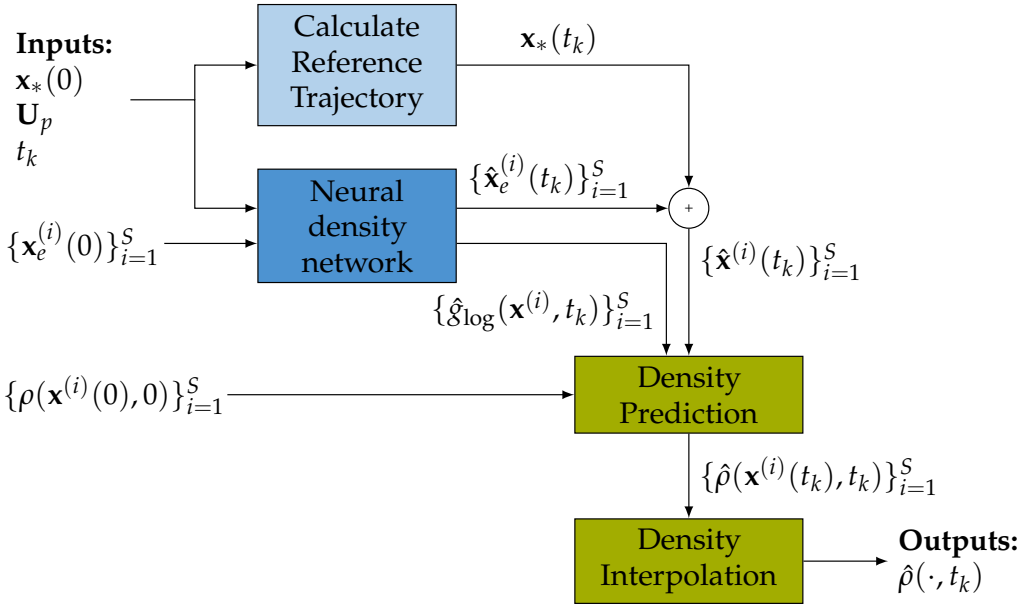



Figure 3.2.: Density prediction process. To predict the density at time $t_k$ around the reference trajectory defined by $\mathbf{x}_*(0)$ and $\mathbf{U}_p$, we first sample $S$ initial states $\mathbf{x}^{(i)}(0)$ from the support of the initial density distribution and calculate the resulting initial deviation $\mathbf{x}_e^{(i)}(0)$. Then, we compute the outputs of the neural network for all input tensors $\{[\mathbf{x}_*(0),\ \mathbf{U}_p,\ t_k,\ \mathbf{x}_e^{(i)}(0)]\}_{i=1}^S$. Finally, the density distribution can be approximated by using Eq. (3.10) on the network outputs and by interpolating and normalizing $\{\hat{\rho}(\mathbf{x}^{(i)}(t_k), t_k)\}_{i=1}^S$ in order to receive a valid probability density distribution.

## 3.3. Computing the Collision Probability

In this thesis, we assume that probabilistic predictions for the evolution of the environment are given. Although we just consider two-dimensional environments, the approach can be easily expanded to higher dimensions.

Furthermore, we use a discrete representation of the environment. Namely, the environment is discretized along its $x$-axis in $C_x$ equally spaced bins and along its $y$-axis in $C_y$ bins resulting in $C_x \cdot C_y$ grid cells which can be indexed by $(c_x, c_y)$. For each grid cell $(c_x, c_y)$ and point in time $t_k$, the probability $P_{\text{occ}}(c_x, c_y, t_k)$ that cell $(c_x, c_y)$ is occupied by an obstacle at time $t_k$ is known. Obstacles could be lane separators, road users like pedestrians, cyclists or other vehicles, or objects like traffic signs or fire hydrants. The occupation probabilities can in general be generated by environment predictors as described in [10, 11, 12] and will be stored in a 3-dimensional tensor where the first two dimensions mark the $x$- and $y$-position of the grid cell and the third dimension symbolizes the time. The discretization of the environment is illustrated in Fig. 3.3.

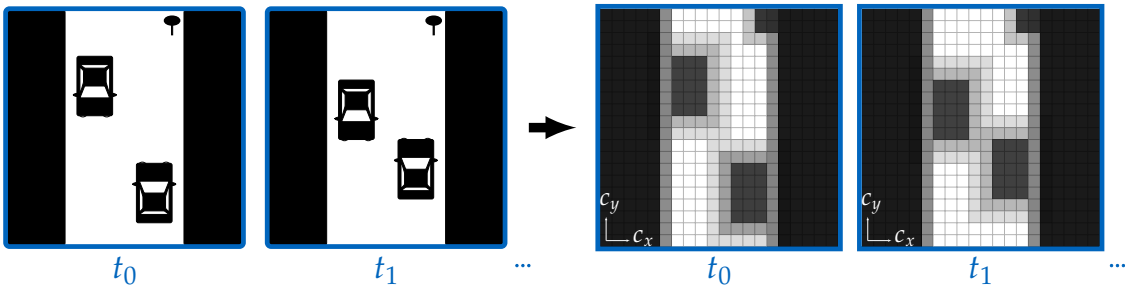



Figure 3.3.: Discretization of the environment: The two left figures show predictions of an example environment for the times $t_0$ and $t_1$. These predictions are usually very uncertain and can be converted to occupation grids where we assign to each grid cell the probability that it is occupied. The occupation probabilities are visualized in the right part of the figures - the darker the grid cell, the higher is its occupation probability.

Next, we compute the density distribution of the position of our ego vehicle as follows:

1. We randomly sample initial states $\{\mathbf{x}^{(i)}(0)\}_{i=1}^{N}$ from the support of the given initial density distribution and compute the initial deviation from the reference trajectory $\{\mathbf{x}_e^{(i)}(0)\}_{i=1}^{N}$.

2. Since we assume in this chapter, that the reference inputs and the reference trajectory is given, we can use the approach from Fig. 3.2 to predict the density $\{\hat{\rho}(\mathbf{x}^{(i)}(t_k), t_k)\}_{i=1}^{N}$.

3. To get a 3-dimensional tensor with the occupation probabilities of the ego vehicle, $P_{\text{ego}}(c_x, c_y, t_k)$, we use the binning approach from [40] where we assign the corresponding grid cell $(c_x^{(i)}, c_y^{(i)})$ in the discretized environment to each predicted sample position $\hat{\mathbf{x}}^{(i)}(t_k)$. The densities of all samples falling into the same grid cell are averaged and the result is stored in the tensor at the corresponding grid cell and time position. Lastly, to get the probability density, the sum over the $x$- and $y$-dimension must be normalized to 1.

To analyze the collision probability for cell $(c_x, c_y)$, both tensors of occupation probabilities are multiplied element-wise:

$$P_{\text{coll}}(c_x, c_y, t_k) = P_{\text{occ}}(c_x, c_y, t_k) \odot P_{\text{ego}}(c_x, c_y, t_k). \tag{3.11}$$

Summing the resulting tensor over the x and y dimension at a certain point in time, gives the overall collision probability at this time:

$$P_{\text{coll}}(t_k) = \sum_{c_x=1}^{C_x} \sum_{c_y=1}^{C_y} P_{\text{coll}}(c_x, c_y, t_k). \tag{3.12}$$

# 4. Optimization of the Reference Trajectory

After being able to compute the collision probability for a given reference trajectory, we now want to find the reference trajectory which minimizes the collision probability while reaching the goal state and satisfying the dynamic and state space constraints. On that account, we create a cost function containing terms for all optimization objectives in Section 4.1. This cost will be minimized with a gradient-based optimization algorithm which is described in Section 4.2.

## 4.1. The Cost Function

The cost function provides a score on the performance of a given input parametrization $\mathbf{U}_p$ and the resulting state and input trajectory $\big(\mathbf{x}(\cdot),\ \mathbf{u}(\cdot)\big)$ with regard to the optimization objectives. Here, we want to minimize collisions, reach the goal, utilize little input actuation and have a state trajectory in the valid state space. Hence, we have four objectives which results in fours terms for our cost function. Each term will be weighted by a factor $\alpha$ according to its importance (e.g., minimizing the control effort is a secondary goal such that its corresponding weighting factor should be small). Thus, we get the following cost function to evaluate trajectory $\big(\mathbf{x}(\cdot),\ \mathbf{u}(\cdot)\big)$:

$$J\big(\mathbf{x}(\cdot),\mathbf{u}(\cdot)\big) = \alpha_{\text{goal}} J_{\text{goal}} + \alpha_{\text{input}} J_{\text{input}} + \alpha_{\text{bounds}} J_{\text{bounds}} + \alpha_{\text{coll}} J_{\text{coll}}. \tag{4.1}$$

The cost terms $J_{\text{goal}}$, $J_{\text{input}}$ and $J_{\text{bounds}}$ will be presented in Section 4.1.1, while Section 4.1.2 explains the computation of $J_{\text{coll}}$. All four terms depend on the trajectory, but the arguments are omitted to improve the readability.

### 4.1.1. Goal, Input and State Space Cost

The first term, $J_{\text{goal}}$, penalizes the distance from the final state of the state trajectory to the goal state, weighted by the final density of the trajectory, and the second term $J_{\text{input}}$ tries to minimize the control effort. They can be calculated with

$$J_{\text{goal}} = \rho(\mathbf{x}(t_N), t_N)\big(\mathbf{x}(t_N) - \mathbf{x}_{\text{goal}}\big)^T \mathbf{Q}_{\text{goal}} \big(\mathbf{x}(t_N) - \mathbf{x}_{\text{goal}}\big) \tag{4.2}$$

$$J_{\text{input}} = \sum_{k=0}^{N-1} \mathbf{u}(t_k)^T \mathbf{Q}_{\text{input}} \mathbf{u}(t_k) \tag{4.3}$$

where $t_N$ is the final time, $\mathbf{x}_{\text{goal}}$ is the goal state which the ego vehicle is supposed to reach and $\mathbf{Q}$ are weighting matrices.

The term $J_{\text{bounds}}$ aims to keep the state trajectory in the valid state space and is computed with

$$J_{\text{bounds}} = \sum_{k=0}^{N} \rho(\mathbf{x}(t_k), t_k) \Big( \big(\mathbf{x}(t_k) - \mathbf{x}_{\min}\big)^T \mathbf{Q}_{\min}(\mathbf{x}(t_k)) \big(\mathbf{x}(t_k) - \mathbf{x}_{\min}\big) + \big(\mathbf{x}(t_k) - \mathbf{x}_{\max}\big)^T \mathbf{Q}_{\max}(\mathbf{x}(t_k)) \big(\mathbf{x}(t_k) - \mathbf{x}_{\max}\big) \Big) \tag{4.4}$$

where $\mathbf{x}_{\min}$ is the minimum and $\mathbf{x}_{\max}$ the maximum bound for the state space, and $\mathbf{Q}_{\min}(\mathbf{x}(t_k))$ and $\mathbf{Q}_{\max}(\mathbf{x}(t_k))$ are diagonal matrices with the diagonal elements

$$\mathbf{Q}_{\min,jj}(\mathbf{x}(t_k)) = \begin{cases} 1 \text{ if } \mathbf{x}_j(t_k) < \mathbf{x}_{\min,j} \\ 0 \text{ else} \end{cases} \quad \text{and} \tag{4.5}$$

$$\mathbf{Q}_{\max,jj}(\mathbf{x}(t_k)) = \begin{cases} 1 \text{ if } \mathbf{x}_j(t_k) > \mathbf{x}_{\max,j} \\ 0 \text{ else} \end{cases} \tag{4.6}$$

respectively, where $\mathbf{x}_j$ denotes the $j$th element of the vector $\mathbf{x}$.

### 4.1.2. Collision Cost

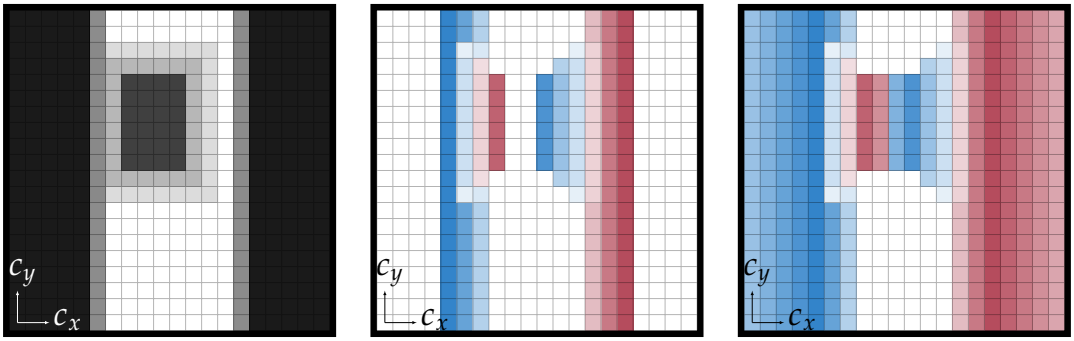
The last term, $J_{\text{coll}}$, describes the cost for high collision probabilities and is more difficult to compute. Due to the non-differentiability of the implemented interpolation and binning method from [40], the collision probability cannot be minimized directly by a gradient-based optimization algorithm. Instead, we compute a differentiable collision cost for $\mathbf{x}(t_k)$ as follows:

1. We compute the direction for each grid cell which leads to a decrease of occupation probability at a given time $t_k$. This can be done by comparing the occupation probabilities of adjoining cells in $x$- and $y$-direction and computing the gradient tensors $G_x$ and $G_y$

$$G_x(c_x, c_y, t_k) = \frac{P_{\text{occ}}(c_x + s, c_y, t_k) - P_{\text{occ}}(c_x - s, c_y, t_k)}{2s} \tag{4.7}$$

$$G_y(c_x, c_y, t_k) = \frac{P_{\text{occ}}(c_x, c_y + s, t_k) - P_{\text{occ}}(c_x, c_y - s, t_k)}{2s} \tag{4.8}$$

   where we use $s = 1$ as step size. The computation of $G_x$ is illustrated in Fig. 4.1b for the example environment from Fig. 4.1a. However, we can see the problem of using a small step size $s$ in Fig. 4.1b: The gradient is zero at the left and right side of the grid map and in the middle of the obstacle where we have areas of constant occupation probabilities. Since the collision probability is nonzero at these cells, we want to have a nonzero gradient which gives us the direction of collision probability decrease. Hence, we have to consider a bigger neighborhood and repeat the gradient computation in these cells, which leads to the procedure




(a) $P_{\text{occ}}(t_k)$ (b) $G_x(t_k)$ after one iteration with $s = 1$ (c) Final $G_x(t_k)$

Figure 4.1.: Visualization of the occupation probability $P_{\text{occ}}$ and the resulting gradient tensor $G_x$ at time $t_k$. The darker the color, the higher the absolute value of the occupation probability or the gradient. Blue cells denote a positive gradient in $x$-direction, red cells a negative gradient and white cells have zero gradient.

depicted in Algorithm 1. Reasonable values for the step size parameters $s_{\text{start}}$ and $s_{\text{diff}}$ are 5 and 10, respectively. The final gradient tensor is visualized in Fig. 4.1c.

This step has to be done only once at the beginning of the optimization procedure and can be computed efficiently by shift functions operating on the whole 3-dimensional occupation probability tensor.

2. Next, we check in which grid cell the sample $\mathbf{x}(t_k)$ lies. If at least one of the corresponding gradients is nonzero, we can compute the desired grid position $(c_{x,\text{des}}(\mathbf{x}(t_k)), c_{y,\text{des}}(\mathbf{x}(t_k)))$ by following the direction of the gradients

$$c_{x,\text{des}}(\mathbf{x}(t_k)) = c_x(\mathbf{x}(t_k)) + \beta G_x(c_x, c_y, t_k) \tag{4.9}$$
$$c_{y,\text{des}}(\mathbf{x}(t_k)) = c_y(\mathbf{x}(t_k)) + \beta G_y(c_x, c_y, t_k) \tag{4.10}$$

where $\beta$ is the step wide and the operators $c_x(\mathbf{x})$ and $c_y(\mathbf{x})$ map the state $\mathbf{x}$ onto its corresponding grid position in $x$- and $y$-direction, respectively.

Next, the desired position in grid coordinates, $\big(c_{x,\text{des}}(\mathbf{x}), c_{y,\text{des}}(\mathbf{x})\big)$, can be transformed to the real-world position $\big(p_{x,\text{des}}(\mathbf{x}), p_{y,\text{des}}(\mathbf{x})\big)$.

3. The collision cost of sample $\mathbf{x}(t_k)$ can now be computed by measuring the squared distance between its $x$- and $y$-position and the desired $x$- and $y$-position. By trying to minimize the squared distance, sample $\mathbf{x}(t_k)$ is pulled to a position with lower occupation probability and the bigger the gradient, the larger is the pull. Additionally, the squared distance gets weighted by the collision probability the sample $\mathbf{x}(t_k)$ induces:

$$P_{\text{coll}}(\mathbf{x}(t_k)) = P_{\text{occ}}\big(c_x(\mathbf{x}(t_k)), c_y(\mathbf{x}(t_k)), t_k\big) \cdot \rho(\mathbf{x}(t_k), t_k) \; . \tag{4.11}$$

**Algorithm 1** Computation of the Gradient Tensors

**function** ComputeGrad($P_{\text{occ}}$)
  **for** $c_x \in \{1, ..., C_x\}$ **do**
    **for** $c_y \in \{1, ..., C_y\}$ **do**
      **for** $k \in \{0, ..., N\}$ **do**
        $s \leftarrow 1$
        $G_x(c_x, c_y, t_k) \leftarrow$ Eq. (4.7)
        $G_y(c_x, c_y, t_k) \leftarrow$ Eq. (4.8)
        $s \leftarrow s_{\text{start}}$
        **while** $G_x(c_x, c_y, t_k) == 0$ and $G_y(c_x, c_y, t_k) == 0$ and
            $P_{\text{occ}}(c_x, c_y, t_k) > 0$ **do**
          $G_{x,new}(c_x, c_y, t_k) \leftarrow$ Eq. (4.7)
          $G_{y,new}(c_x, c_y, t_k) \leftarrow$ Eq. (4.8)
          $G_x(c_x, c_y, t_k) \leftarrow G_x(c_x, c_y, t_k) + s \cdot G_{x,new}(c_x, c_y, t_k)$
          $G_y(c_x, c_y, t_k) \leftarrow G_y(c_x, c_y, t_k) + s \cdot G_{y,new}(c_x, c_y, t_k)$
          $s \leftarrow s + s_{\text{diff}}$
  **return** $G_x, G_y$

Consequently, the overall collision cost for the trajectory $\mathbf{x}(\cdot)$ can be calculated with

$$J_{\text{coll}} = \sum_{k=0}^{N} P_{\text{coll}}(\mathbf{x}(t_k)) \Big( \big(p_x(\mathbf{x}(t_k)) - p_{x,\text{des}}(\mathbf{x}(t_k))\big)^2 + \big(p_y(\mathbf{x}(t_k)) - p_{y,\text{des}}(\mathbf{x}(t_k))\big)^2 \Big). \tag{4.12}$$

This cost will be termed collision risk from now on. The procedure for computing the collision risk is summarized in Algorithm 2.

Finally, the overall cost can be minimized, e.g., by gradient descent or with the ADAM optimizer [45].

## 4.2. The Optimization Approach

In the planning phase, we want to find the optimal reference trajectory after having received a new environment setup (initial distribution, goal state and the environment prediction). In order to keep the planning time small, the optimization approach should have low computational complexity. Furthermore, we want to overcome bad local optima of the cost function presented in the previous section, which leads to the following two step procedure: First, we generate a certain number of random input trajectories, which are optimized without considering the uncertain initial state. This step is explained in Section 4.2.1. The trajectory which has the lowest final cost is chosen to be optimized with the density predictions. Section 4.2.2 presents this part in detail.

**Algorithm 2** Computation of the Collision Risk

**function** ComputeCollRisk($P_{\text{occ}}$, $G_x$, $G_y$, $\mathbf{x}(\cdot)$, $\rho(\cdot)$)
  **for** $k \in \{0, ..., N\}$ **do**
    $p_x(t_k),\ p_y(t_k) \leftarrow$ get position of sample $\mathbf{x}(t_k)$ in real-world coordinates
    $c_x(t_k),\ c_y(t_k) \leftarrow$ transform $(p_x(t_k),\ p_y(t_k))$ to grid coordinates
    $c_{x,\text{des}}(t_k) \leftarrow$ Eq. (4.9)
    $c_{y,\text{des}}(t_k) \leftarrow$ Eq. (4.10)
    $p_{x,\text{des}}(t_k),\ p_{y,\text{des}}(t_k) \leftarrow$ transform $(c_{x,\text{des}}(t_k),\ c_{y,\text{des}}(t_k))$ to position in real-world coordinates
    $P_{\text{coll}}(t_k) \leftarrow$ Eq. (4.11)
  $J_{\text{coll}} \leftarrow$ Eq. (4.12)
  **return** $J_{\text{coll}}$

### 4.2.1. Initialization

As the optimization with the whole density distribution is computationally complex and can get easily stuck in local minima of the cost landscape, we want to find a good initial guess first. This can be achieved by discretizing the input parameter space and using search methods. Since the performance of a search approach depends massively on the discretization (large discretization steps easily overlook good trajectories while small steps lead to long computation times), we rather choose a gradient-based optimization approach which is depicted in Algorithm 3 and shortly explained in the following. Two other optimization methods are presented in Section 5.3 where an ablation study is conducted.

For the gradient-based approach, we randomly sample $M$ parameter sets $\{\mathbf{U}_p^{(i)}\}_{i=1}^M$ and generate the corresponding reference input trajectories $\{\mathbf{u}_*^{(i)}(\cdot)\}_{i=1}^M$ with Eq. (3.8). Next, we compute the reference state trajectories $\{\mathbf{x}_*^{(i)}(\cdot)\}_{i=1}^M$ by integrating the dynamics Eq. (3.1) starting from the initial state $\mathbf{x}_*(0)$. The initial state can be chosen as the mean of the given initial density distribution and is the same for all $M$ trajectories. Now, we calculate the cost $J\big(\mathbf{u}_*^{(i)}(\cdot), \mathbf{x}_*^{(i)}(\cdot)\big)$ for each trajectory and update the input parameters $\{\mathbf{U}_p^{(i)}\}_{i=1}^M$ with gradient descent until a certain number of iterations is reached. For computing the different cost terms, we do not consider the initial state uncertainty and hence, assume that the density of each state is one.

Since we first want to guide the trajectories to the goal, we initialize $\alpha_{\text{bounds}}$ and $\alpha_{\text{coll}}$ in Eq. (4.1) for all trajectories with zero. At each optimization iteration and for each trajectory pair, we check if the distance to the goal is smaller than a certain threshold. If this condition is fulfilled for one trajectory, the corresponding $\alpha_{\text{bounds}}$ is set to a nonzero value such that the state space constraints get enforced. For all trajectories where $\alpha_{\text{bounds}} \neq 0$, we do another check: If the state space constraints are met, the collision cost will be considered by setting $\alpha_{\text{coll}}$ to a nonzero value. This cost calculation procedure ensures that the trajectories are first steered to the goal, are next moved to the valid

**Algorithm 3** Finding a Good Initial Trajectory

**function** FINDINITIALTRAJECTORY($P_{\text{occ}}$, $G_x$, $G_y$, $\mathbf{x}_*(0)$, $\mathbf{x}_{\text{goal}}$)
  $\{\mathbf{U}_p^{(i)}\}_{i=1}^{M} \leftarrow$ randomly sampled from admissible input parameter space
  **for** $i \in \{1, ..., M\}$ **do** simultaneously
    $\rho^{(i)}(\cdot) \leftarrow \mathbf{1}$
    $\alpha_{\text{bounds}} \leftarrow 0$
    $\alpha_{\text{coll}} \leftarrow 0$
    $j \leftarrow 0$
    **while** $j < j_{\max}$ **do**
      **if** $j > 0$ **then**
        $\mathbf{U}_{p,i} \leftarrow$ update with gradient descent on $J^{(i)}$
      $\mathbf{u}^{(i)}(\cdot) \leftarrow$ Eq. (3.8) with $\mathbf{U}_p^{(i)}$
      $\mathbf{x}^{(i)}(\cdot) \leftarrow$ integration of Eq. (3.1) with $\mathbf{u}^{(i)}(\cdot)$ and starting at $\mathbf{x}_*(0)$
      $J_{\text{goal}} \leftarrow$ Eq. (4.2)
      $J_{\text{input}} \leftarrow$ Eq. (4.3)
      $J_{\text{bounds}} \leftarrow 0$
      $J_{\text{coll}} \leftarrow 0$
      **if** $\alpha_{\text{bounds}} == 0$ and $J_{\text{goal}}$ sufficient small **then**
        $\alpha_{\text{bounds}} \leftarrow$ nonzero value
      **if** $\alpha_{\text{bounds}} > 0$ **then**
        $J_{\text{bounds}} \leftarrow$ Eq. (4.4)
        **if** $\alpha_{\text{coll}} == 0$ and $J_{\text{bounds}}$ sufficient small **then**
          $\alpha_{\text{coll}} \leftarrow$ nonzero value
        **if** $\alpha_{\text{bounds}} > 0$ **then**
          $J_{\text{coll}} \leftarrow$ COMPUTECOLLRISK($P_{\text{occ}}$, $G_x$, $G_y$, $\mathbf{x}^{(i)}(\cdot)$, $\rho^{(i)}(\cdot)$)
      $J^{(i)} \leftarrow$ Eq. (4.1)
  $i_{\min} \leftarrow \arg\min_i J^{(i)}$
  **return** $\mathbf{U}_p^{(i_{\min})}$

state space and only then get optimized with regard to collisions. This procedure saves computation effort and leads to good trajectories much faster than when considering all cost terms from the beginning.

For computational efficiency, we parallelize the algorithm and optimize all input parameter sets $\{\mathbf{U}_p^{(i)}\}_{i=1}^{M}$ simultaneously. Finally, we compare the cost of all optimized parameter sets and return the set with the lowest cost.

### 4.2.2. Local Optimization with Density Predictions

In the next step, we locally optimize the best parameter set from Section 4.2.1 by taking the initial state uncertainties into our consideration. The proposed approach is shown

**Algorithm 4** Local Density Optimization

**function** OPTIMIZETRAJECTORY($P_{\text{occ}}$, $G_x$, $G_y$, $\mathbf{x}_*(0)$, $\mathbf{x}_{\text{goal}}$, $\mathbf{U}_p$, $\rho(\cdot, 0)$)
  $\{\mathbf{x}^{(i)}(0)\}_{i=1}^{S} \leftarrow$ randomly sampled from the support of $\rho(\cdot, 0)$
  $\forall i: \ \mathbf{x}_e^{(i)}(0) \leftarrow \mathbf{x}^{(i)}(0) - \mathbf{x}_*(0)$
  $\forall i: \ \rho^{(i)}(0) \leftarrow \rho(\mathbf{x}^{(i)}(0), 0)$
  $j \leftarrow 0$
  **while** $j < j_{\max}$ **do**
    **if** $j > 0$ **then**
      $\mathbf{U}_p \leftarrow$ update with gradient descent on $J$
    $\mathbf{u}_*(\cdot) \leftarrow$ Eq. (3.8) with $\mathbf{U}_p$
    $\mathbf{x}_*(\cdot) \leftarrow$ integration of Eq. (3.1) with $\mathbf{u}_*(\cdot)$ and starting at $\mathbf{x}_*(0)$
    **for** $i \in \{1, ..., S\}$ **do** simultaneously
      $\mathbf{x}_e^{(i)}(\cdot), \ g_{\log}^{(i)}(\cdot) \leftarrow$ predict with density neural network
      $\rho^{(i)}(\cdot) \leftarrow$ Eq. (3.10) and normalize over all samples
      $\mathbf{x}^{(i)}(\cdot) \leftarrow \mathbf{x}_e^{(i)}(\cdot) + \mathbf{x}_*(\cdot)$
      $J_{\text{goal}} \leftarrow$ Eq. (4.2)
      $J_{\text{input}} \leftarrow$ Eq. (4.3)
      $J_{\text{bounds}} \leftarrow$ Eq. (4.4)
      $J_{\text{coll}} \leftarrow$ COMPUTECOLLRISK($P_{\text{occ}}$, $G_x$, $G_y$, $\mathbf{x}^{(i)}(\cdot)$, $\rho^{(i)}(\cdot)$)
      $J^{(i)} \leftarrow$ Eq. (4.1)
    $J \leftarrow \sum_{i=1}^{S} J^{(i)}$
  **return** $\mathbf{U}_p$

in Algorithm 4.

First, we randomly sample $S$ initial states $\{\mathbf{x}^{(i)}(0)\}_{i=1}^{S}$ from the given initial density distribution $\rho(\cdot, 0)$. Starting from the previously optimized parameter set $\mathbf{U}_p$ from Section 4.2.1, we compute the corresponding reference input trajectory $\mathbf{u}_*(\cdot)$ and reference state trajectory $\mathbf{x}_*(\cdot)$. We use the input parameters $\mathbf{U}_p$, the initial deviation of the sampled states from the reference trajectory $\mathbf{x}_e^{(i)}(0)$, the initial reference state $\mathbf{x}_*(0)$ and the initial density of the sample points $\rho(\mathbf{x}^{(i)}(0), 0)$ as an input for the neural density predictor from Section 3.2, and approximate the state trajectories $\mathbf{x}^{(i)}(\cdot)$ as well as their density $\rho(\mathbf{x}^{(i)}(\cdot), \cdot)$. Next, we can compute the costs $J^{(i)}$ of all reference input trajectory - sample trajectory pairs $(\mathbf{u}_*(\cdot), \mathbf{x}^{(i)}(\cdot))$. The cost computation can be done simultaneously for all samples. If the maximum number of optimization iterations is not reached, we calculate the gradient of the overall cost $\sum_{i=1}^{S} J^{(i)}$ and optimize the input parameters.

By doing so, this approach is similar to the initialization procedure from Section 4.2.1 with the following differences:

- We optimize one set of input parameters, but we consider $S$ possible state trajectories resulting from the uncertain initial state, whereas in Section 4.2.1, we

simultaneously optimize $M$ parameters sets but just consider the reference trajectory for each set.

- We use all cost terms of the cost function from the beginning. As we start from a good initial guess, we can assume that the state trajectories already lead to the goal and lie in the valid state space at the beginning.
- We predict the density evolution for each sample and use it for computing the cost terms.

The overall optimization procedure is depicted in Algorithm 5.

**Algorithm 5** Optimization Procedure

1: **function** PLANTRAJECTORY($P_{\text{occ}}$, $\rho(\cdot, 0)$, $\mathbf{x}_{\text{goal}}$)
2: $\mathbf{x}_*(0) \leftarrow$ mean of $\rho(\cdot, 0)$
3: $G_x, G_y \leftarrow$ COMPUTEGRAD($P_{\text{occ}}$)
4: $U_{p,\text{init}} \leftarrow$ FINDINITIALTRAJECTORY($P_{\text{occ}}$, $G_x$, $G_y$, $\mathbf{x}_*(0)$, $\mathbf{x}_{\text{goal}}$)
5: $U_{p,\text{opt}} \leftarrow$ OPTIMIZETRAJECTORY($P_{\text{occ}}$, $G_x$, $G_y$, $\mathbf{x}_*(0)$, $\mathbf{x}_{\text{goal}}$, $\mathbf{U}_{p,\text{init}}$, $\rho(\cdot, 0)$)
6: **return** $\mathbf{U}_{p,\text{opt}}$

# 5. Application to Autonomous Cars

While the motion planning approach can be applied to all kind of autonomous systems, it will be illustrated here for the example of self-driving cars.

First, the vehicle model will be described in Section 5.1. Next, details for implementing the contraction controller, the neural density predictor and the optimization method are presented in Section 5.2 and their performance is analyzed. In Section 5.3, we compare the gradient-based optimization method with a search-based and a sampling-based approach in large number of simulated environments.

## 5.1. Dubins' Car Model

The system will be described by a simple kinematic car model which was first characterized by Dubins in [46]. It can be controlled by the two inputs $\mathbf{u}(t) = [\omega(t), a(t)]^T$ where $\omega(t)$ is the angular velocity and $a(t)$ is the longitudinal acceleration, and its states are given by

$$\dot{p}_x(t) = v(t)\ \cos(\theta(t)) \tag{5.1}$$
$$\dot{p}_y(t) = v(t)\ \sin(\theta(t)) \tag{5.2}$$
$$\dot{\theta}(t) = \omega(t) \tag{5.3}$$
$$\dot{v}(t) = a(t), \tag{5.4}$$

where $p_x(t)$ is the position along the $x$-axis in a global coordinate system, $p_y(t)$ is the position along the $y$-axis, $\theta(t)$ is the heading angle and $v(t)$ is the longitudinal velocity of the vehicle at time $t$.

As we usually do not have a perfect model or completely precise measurements, we want to demonstrate that the controller can also cope with disturbances. However, the LE which is used to generate the training data cannot be used to compute the density for systems with stochastic disturbances - for this case, we would have to use the FPE which is much more difficult to solve. To be able to use the LE, we have to consider deterministic disturbances. Hence, we choose to implement a constant sensor bias $\theta_{\text{bias}}$ for measuring the heading angle which leads to the augmented state

$\mathbf{x}(t) = [p_x(t),\ p_y(t),\ \theta(t),\ v(t),\ \theta_{\text{bias}}(t)]^T$ and the dynamics

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{p}_x(t) \\ \dot{p}_y(t) \\ \dot{\theta}(t) \\ \dot{v}(t) \\ \dot{\theta}_{\text{bias}}(t) \end{bmatrix} = \begin{bmatrix} v(t)\ \cos(\theta(t)) \\ v(t)\ \sin(\theta(t)) \\ \omega(t) \\ a(t) \\ 0 \end{bmatrix}. \tag{5.5}$$

For the targeted motion planning problem, we want to compute a controller which allows the system to converge to and track a given reference trajectory $\mathbf{x}_*(t)$ independent of the uncertain initial state. Since the dynamics can be written in the input-affine form

$$\dot{\mathbf{x}}(t) = \underbrace{\begin{bmatrix} v(t)\ \cos(\theta(t)) \\ v(t)\ \sin(\theta(t)) \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{a(\mathbf{x})} + \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}}_{B(\mathbf{x})} \underbrace{\begin{bmatrix} \omega(t) \\ a(t) \end{bmatrix}}_{\mathbf{u}(t)}, \tag{5.6}$$

the contraction theory presented in Section 2.2 can be used for computing a tracking controller $\mathbf{u}(t) = \mathbf{u}(\hat{\mathbf{x}}(t), \mathbf{x}_*(t), \mathbf{u}_*(t))$ where $\hat{\mathbf{x}}(t)$ denotes the measured state

$$\hat{\mathbf{x}}(t) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{x}(t) \tag{5.7}$$

and $\mathbf{x}_*(t)$ is the reference state, i.e.,

$$\mathbf{x}_*(t) = [p_{x*}(t),\ p_{y*}(t),\ \theta_*(t),\ v_*(t),\ 0]^T. \tag{5.8}$$

## 5.2. Implementation

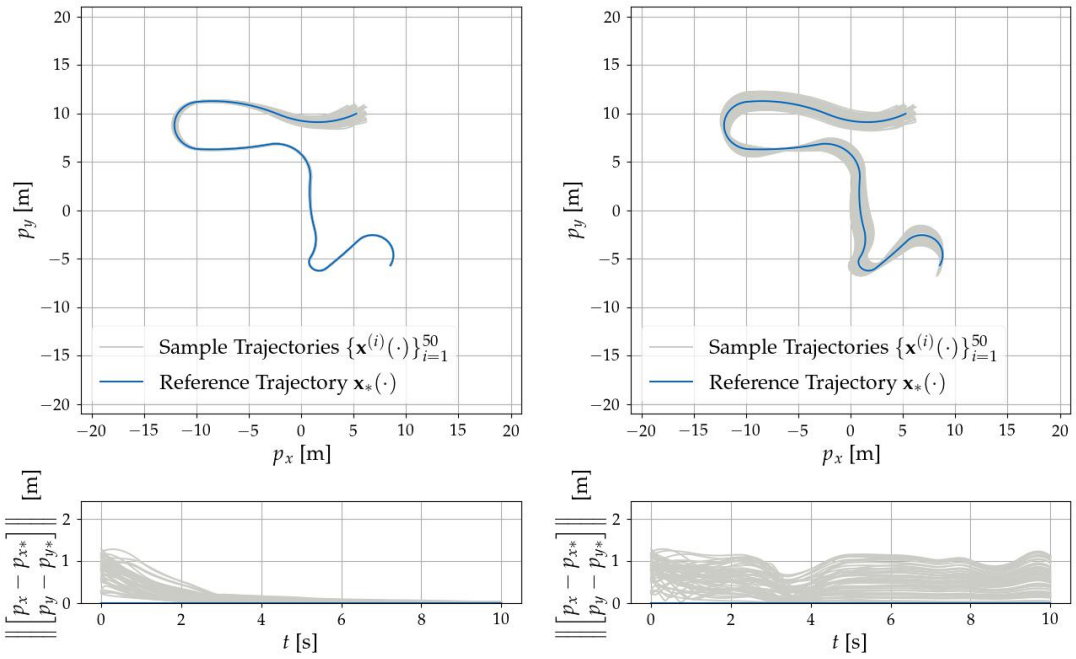
This section describes the implementation of the proposed motion planning algorithm applied to Dubins' Car Model. First, the controller for the car dynamics is synthesized and its performance is visualized. Furthermore, we analyze the neural density predictor and evaluate the optimization method.

### 5.2.1. Contraction Controller

To track the reference trajectory, we implement the contraction controller from Section 3.1. In [43], the matrices $\mathbf{N}_2$ and $\mathbf{N}_3$ of the controller (see Eq. (3.2)) are approximated by two 2-layer neural networks with 128 hidden neurons which leads to a small training loss

in the considered state space. However, to be able to solve a large number of motion planning tasks, we have to extend the state and input space. The bounds of the original and of the new state space are presented in Table A.1 in the appendix. Since the original 2-layer controller leads to a high training loss in the extended state space, we increase the size of the neural networks. For motion planning, we will use a controller with 3-layer neural networks which can significantly decrease the training loss.

To evaluate its tracking performance, we apply the contraction controller to a large number of states and reference trajectories. The resulting tracking errors for one example reference trajectory are visualized in Fig. 5.1. Fig. 5.1a shows the case of perfect measurements where all trajectories converge very fast to the reference trajectory. In Fig. 5.1b the controller is not able to measure the heading angle exactly because of sensor bias and hence, complete convergence to the reference trajectory cannot be achieved. Nevertheless, the contraction controller is still able to keep the trajectories in the vicinity of the reference which is desirable for motion planning.




(a) Perfect measurements.

(b) With sensor bias.

Figure 5.1.: Tracking performance of the neural contraction controller. First, a reference trajectory is randomly generated and 50 initial states are sampled from the initial density distribution. Then, the resulting state trajectories can be computed by applying the tracking controller. The figures at the top show the trajectories in the $x$-$y$ plane, while the time curve of the tracking error is given below.

### 5.2.2. Neural Density Predictor

Next, we train a neural network to predict the density distribution along the reference trajectory. For this, we use the descriptions from Section 3.2 to generate the training data for Dubins' car dynamics and parameterize the reference input trajectory by the matrix $\mathbf{U}_p$ such that

$$\mathbf{u}_*(t) = \begin{cases} \mathbf{U}_{p:1} & \text{if } 0\,\text{s} \leq t < 1\,\text{s} \\ \mathbf{U}_{p:2} & \text{if } 1\,\text{s} \leq t < 2\,\text{s} \\ \dots & \\ \mathbf{U}_{p:10} & \text{if } 9\,\text{s} \leq t < 10\,\text{s} \end{cases}, \tag{5.9}$$

where $\mathbf{U}_{p:i}$ is the $i$th column of $\mathbf{U}_p$. After comparing the loss curves of different hyperparameter configurations, we decide on an architecture with seven layers and 150 neurons. Furthermore, we choose the factor $\alpha_g$ in the cost function Eq. (3.9) to be quite small such that we have similar magnitudes for the density error $\mathscr{L}_g$ and the state prediction error $\mathscr{L}_\mathbf{x}$. The loss curve for the final hyperparmater configuration is displayed in Fig. 5.2.

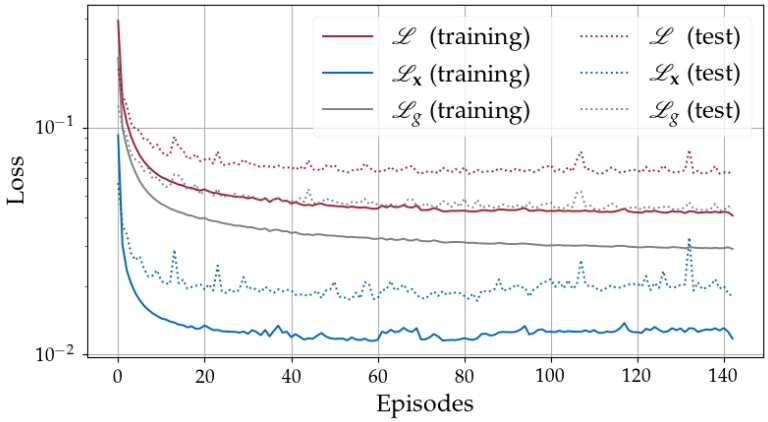
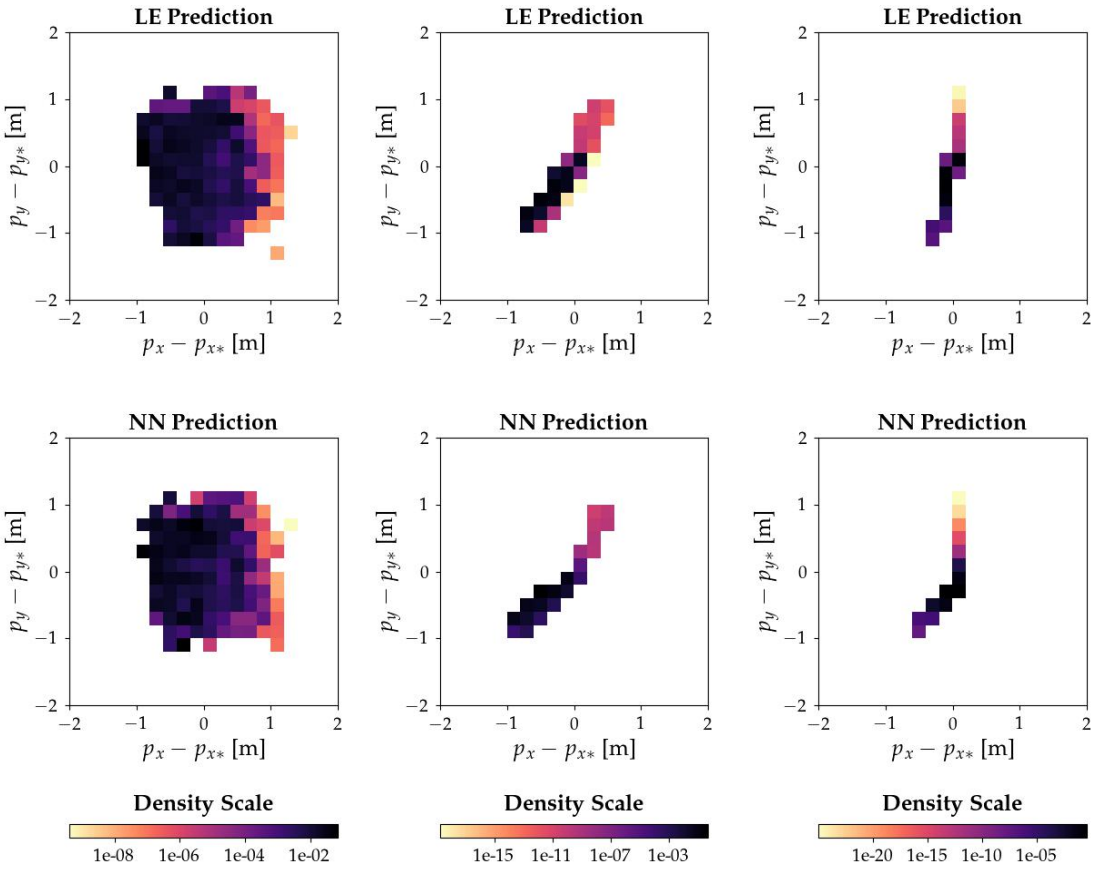



Figure 5.2.: Final loss curve for the training of the neural density predictor with $\alpha_g = 0.0005$.

Although, the loss for predicting the logarithmic density concentration function is still quite high, the predictions for the whole density distribution are similar to the results of the LE. This is visualized in Fig. 5.3. Hence, the density network provides acceptable approximations of the density distribution. Additionally, the computation times are very small: While the prediction of the 10 s-state and -density trajectories starting from 500 initial states takes on average 11.6 s when integrating the system dynamics and using the Liouville equation, the density network decreases this time to 0.3 s. The numerical data is provided in Appendix B.2.

Consequently, the network can be used to accelerate the computation of the density and state trajectories in our motion planning approach.




(a) Prediction for $t_k = 0.5\,s$. (b) Prediction for $t_k = 5\,s$. (c) Prediction for $t_k = 10\,s$.

Figure 5.3.: Performance of the neural density predictor. To generate the heatmap plots, we sample 1000 initial states from the support of the given density distribution (here, we assume an uniform distribution with a cuboid support) and predict their density for a random reference trajectory with the Liouville equation as ground truth and with the trained neural network (here, we use the same reference trajectory as in Fig. 5.1). Finally, we interpolate and normalize the results.

### 5.2.3. Trajectory Optimization

Lastly, we implement the algorithm from Chapter 4 to plan the reference trajectory. As we only want to reach a certain final position but do not have any requests for the final velocity and heading angle, we use the following weighting matrix for the goal cost:

$$\mathbf{Q}_{\text{goal}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

As weighting matrix $\mathbf{Q}_{\text{input}}$ we use the identity matrix and the cost factors $\alpha$ are chosen in such a way that the resulting overall cost gradient is very sensible to $J_{\text{goal}}$ and $J_{\text{coll}}$ but is dominated by $J_{\text{bound}}$ if the trajectory is out-of-bounds. The values are displayed in Appendix A.2. Furthermore, we clip the gradient if its absolute value becomes too big, to avoid large jumps during the optimization.

The costs which result from the trajectory optimization for an example motion planning task are illustrated in Fig. 5.4.

In Fig. 5.4a, we can see that the collision risk is considered (i.e., $\alpha_{\text{coll}}$ is set to a nonzero value) after 24 iterations in the initialization procedure, as the goal cost has become sufficiently small. During the density optimization procedure in Fig. 5.4b, all cost terms are included from the beginning. Since the trajectory does not violate any state space constraints during the whole optimization procedure, $J_{\text{bounds}}$ is zero at every iteration.

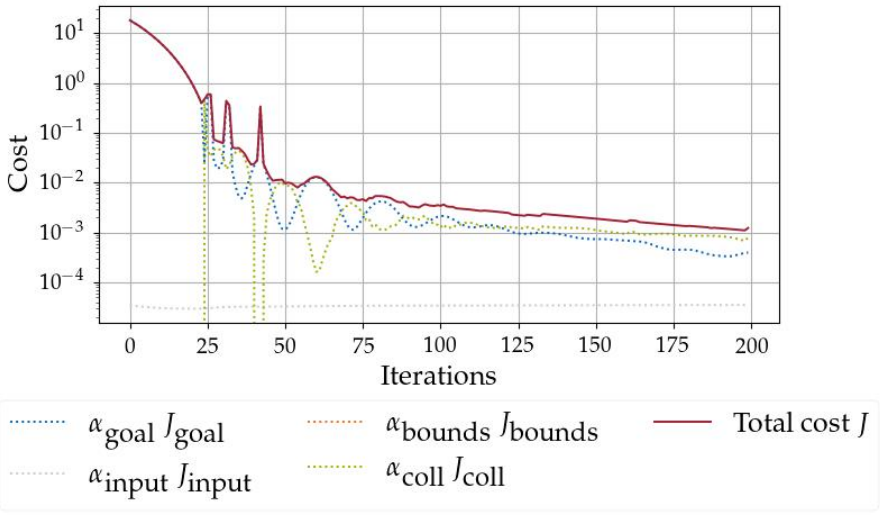
Both curves show that the costs are consistently decreased and hence, the optimization method is effective. However, the slope shrinks and after 100 iterations only minor improvements are achieved. Hence, to minimize the planning time we can set the maximum number of iteration to 100.

A comparison with other optimization methods as well as an analysis of its average computation time is provided in the next section.

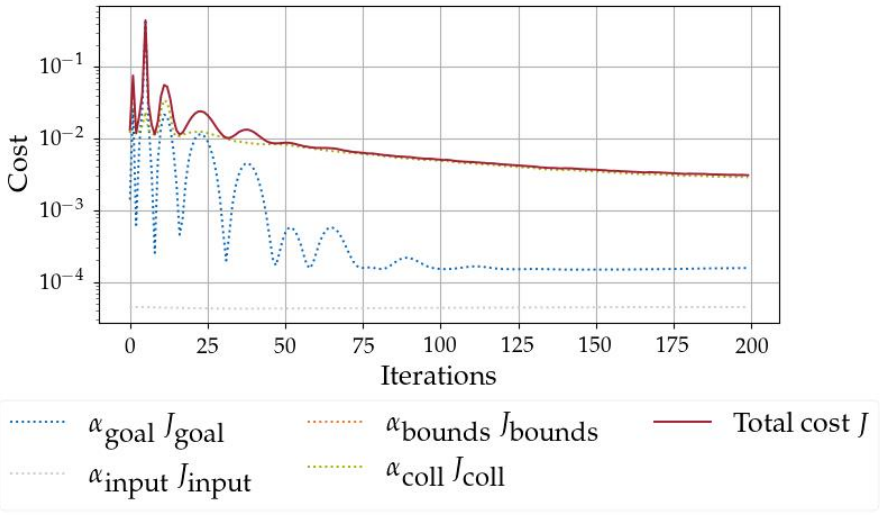
## 5.3. Ablation Study for the Optimization Method

To evaluate the performance, we compare the proposed gradient-based optimization method with a sampling-based and a search-based approach. We use the same solution concept, i.e., we plan a reference trajectory which is tracked by a contraction controller and utilize the same neural density predictor. Just the method for the trajectory optimization will be exchanged.

After introducing the two alternative optimization approaches in Section 5.3.1 and Section 5.3.2, we show the superiority of our method in a large number of simulated environments in Section 5.3.3.




(a) Example cost curve of the initialization procedure.




(b) Cost curve of the density optimization procedure.

Figure 5.4.: Cost curves of the gradient-based optimization method. First, we optimize 100 random trajectories with Algorithm 3 (initialization procedure). Then, the best one gets optimized with Algorithm 4 (density optimization procedure).

### 5.3.1. Search-based Trajectory Optimization

For the search-based method, we investigate a greedy best-first search in the parameter space $\mathbf{U}_p = [\mathbf{U}_{p:1}, \mathbf{U}_{p:2}, ..., \mathbf{U}_{p:10}]$ where each vector $\mathbf{U}_{p:i}$ can take the following values:

$$\mathbf{U}_{p:i} \in \{ \begin{bmatrix} k_1 \cdot \Delta\mathbf{u}_1 \\ k_2 \cdot \Delta\mathbf{u}_2 \end{bmatrix} \Bigg| \ \mathbf{u}_{\min} \leq \begin{bmatrix} k_1 \cdot \Delta\mathbf{u}_1 \\ k_2 \cdot \Delta\mathbf{u}_2 \end{bmatrix} \leq \mathbf{u}_{\max},\ k_1, k_2 \in \mathbb{Z} \}, \quad \forall i \in \{1, 2, ..., 10\}, \quad (5.10)$$

and $\Delta\mathbf{u} = [\Delta\mathbf{u}_1,\ \Delta\mathbf{u}_2]^T$ is the discretization step.

To start the search, we evaluate the cost for all possible values of $\mathbf{U}_p = [\mathbf{U}_{p:1}]$. This is done by recovering the input trajectory for the first second with Eq. (5.9) and computing the first segment of the reference trajectory. Then, the cost of each of these short trajectories is calculated with

$$J_{\text{search}} = \alpha_{\text{goal}} J_{\text{goal}} + \alpha_{\text{coll}} J_{\text{coll}}, \quad (5.11)$$

where $J_{\text{goal}}$ and $J_{\text{coll}}$ is computed with the algorithm from Section 4.1. Furthermore, we check if the trajectories lie in the valid state space. Finally, we add all investigated values of $\mathbf{U}_p = [\mathbf{U}_{p:1}]$, for which the state constraints are fulfilled, together with their costs to the frontier.

For the next search iterations, we remove the node with the lowest cost, $\mathbf{U}_{p,\text{best}}$, from the frontier and expand it with all possible values of $\mathbf{U}_{p:i}$ from Eq. (5.10) where $i$ is the number of columns of $\mathbf{U}_{p,\text{best}}$ plus one. We compute the costs for the resulting trajectories described by the extended $\mathbf{U}_p = [\mathbf{U}_{p,\text{best}}, \mathbf{U}_{p:i}]$ and add the valid ones to the frontier. This process is repeated until a trajectory with the length of 10 s is found (i.e., the node $\mathbf{U}_{p,\text{best}}$ consists of ten columns). If this trajectory ends close to the goal position, we accept the trajectory as the solution of the search. If the distance to the goal is bigger than a certain threshold, we remove the trajectory from the frontier and continue with the search. How the search tree could look like after four search iterations is visualized in Fig. 5.5.

By tuning the parameters $\alpha_{\text{goal}}$ and $\alpha_{\text{coll}}$ of the cost function, we can adjust the weighting of quickly reaching the goal versus emphasizing the minimization of the collision probability in the search process. If $\alpha_{\text{goal}}$ is too small, the search will focus on trajectories with low collision probability, but it will take a long time until a trajectory leading to the goal will be found.

In addition, we implement two heuristics to avoid expanding unpromising trajectories and hence save computation time. For this, we compute the heading angle which would be necessary to reach the goal position from the end of the reference trajectory in a straight line and compare it with the actual heading angle of the last state. If the difference between both angles is bigger than a threshold, we assume that the trajectory does not lead to the goal and remove it from the frontier. Similarly, we calculate the average velocity which would be necessary to reach the goal at the final time $t_N = 10\,s$ starting from the last state of the reference trajectory. If this velocity is larger than the maximum possible velocity, we know that the trajectory is not able to reach the goal in time and we remove it from the frontier.

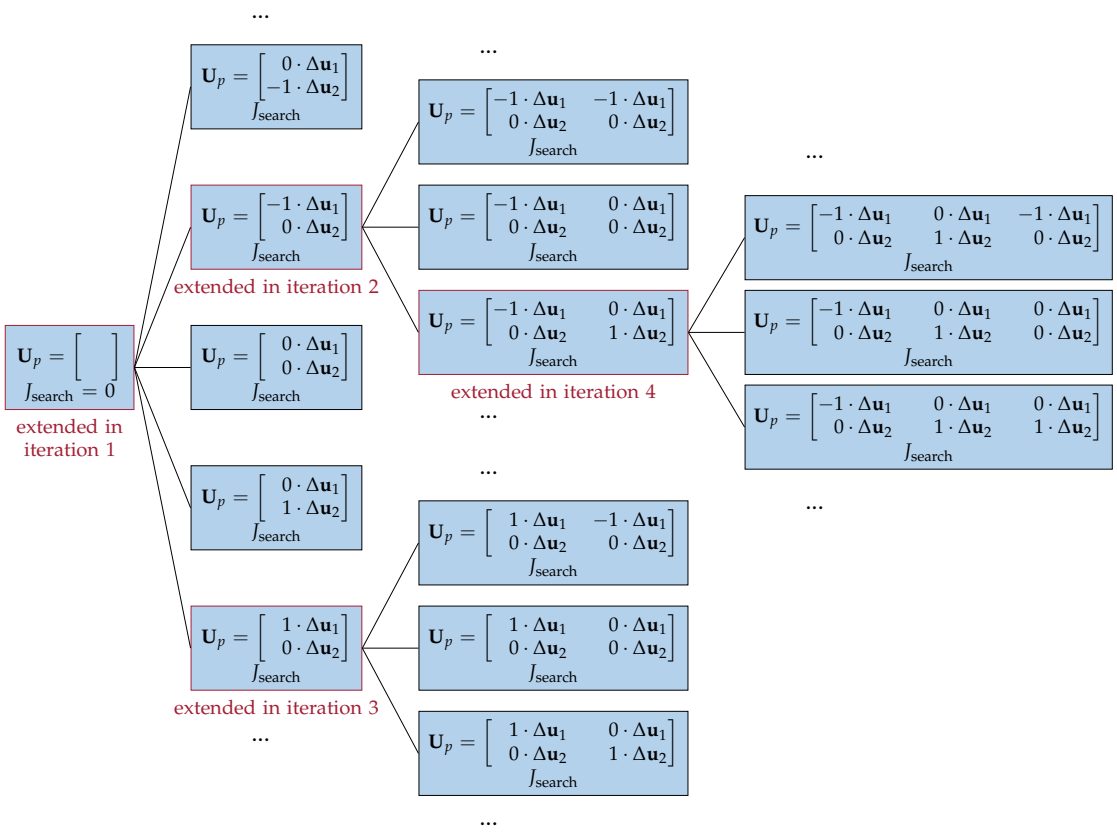



Figure 5.5.: Example search tree after four iterations. The node with the lowest cost at the current iteration (framed in red) is extended and then removed from the frontier.

Next, we have to tune the discretization step $\Delta\mathbf{u}$ which determines the size of the search space and consequently has a big influence on the computation time. While $\Delta\mathbf{u} = [0.5,\ 0.5]^T$ leads to intractable long computation times, the step $\Delta\mathbf{u} = [2,\ 2]^T$ is too coarse such that trajectories leading to the vicinity of the goal cannot be found in many environments. However, the search algorithm is able to find a valid solution with $\Delta\mathbf{u} = [1,\ 2]^T$ (we choose the discretization step for the input $\omega$ to be smaller than for $a$ since we assume that the collision probability of a trajectory is more sensible to $\omega$) in six to eight minutes. By constraining the input parameters to be pairwise equal, i.e., $\mathbf{U}_{p:i} = \mathbf{U}_{p:i+1}\ \forall i \in \{1,3,5,7,9\}$, we are able to find solutions in about the same time for $\Delta\mathbf{u} = [1,\ 1]^T$.

### 5.3.2. Sampling-based Trajectory Optimization

As the performance of the search-based method is severely limited by the discretization of the possible values of $\mathbf{U}_{p:i}$, we also implement a more flexible sampling-based method where $\mathbf{U}_{p:i}$ can take every value in the given input bounds.

At the beginning, we randomly sample $\mathbf{U}_{p:1}$ from a Gaussian distribution with mean $[0,\ 0]$ and clip it at the input bounds, such that

$$\mathbf{U}_{p:1} \in \left\{ \mathbf{U}_{p:i} | \mathbf{u}_{\min} \leq \mathbf{U}_{p:i} \leq \mathbf{u}_{\max} \right\} . \tag{5.12}$$

Next, we evaluate the cost for the trajectory described by $\mathbf{U}_p = [\mathbf{U}_{p:1}]$ with the cost function from Eq. (5.11). The trajectory and its cost is saved if the trajectory fulfills the state space constraints as well as the heading angle and the velocity heuristic from the previous subsection.

In the next iterations, we first randomly decide if we want to start a new trajectory by sampling $\mathbf{U}_{p:1}$ whereas the probability for starting a new trajectory decreases with the number of saved trajectories. If we decide against a new trajectory, an already saved trajectory will be extended with a randomly sampled $\mathbf{U}_{p:i}$. Which trajectory gets extended is chosen randomly where the probability for each trajectory depends on its cost and the number of times it has been extended previously, i.e., a trajectory with low cost and which has not been extended before has a higher chance of being chosen. Trajectories which have reached the maximum duration of 10 s will not be extended anymore.

The sampling and extension process is terminated as soon as a 10 s-trajectory is found which has a low collision probability and reaches the vicinity of the goal.

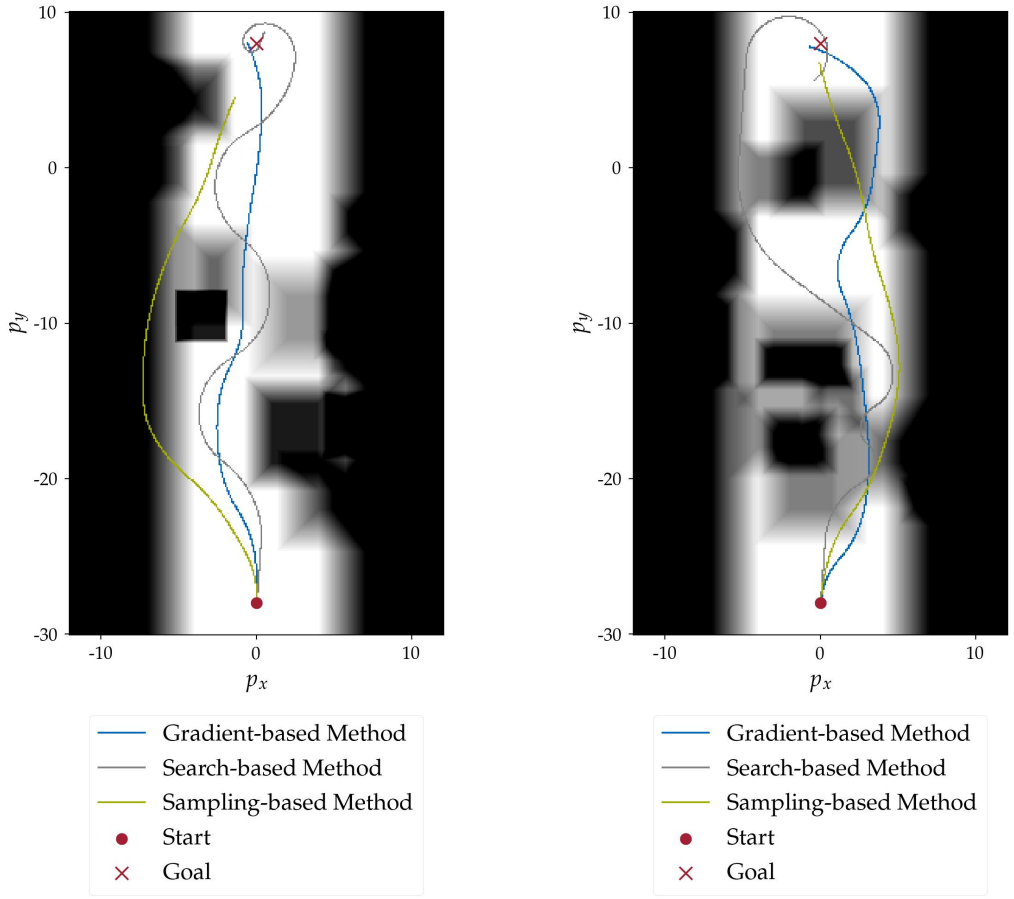
### 5.3.3. Comparison of the Optimization Methods

Lastly, we want to compare the performance of the three optimization approaches in terms of computation time and final cost. For this, we generate a large number of simulated environments, i.e., streets with a random number of uncertain stationary and dynamic obstacles. The obstacle sizes, positions, uncertainties and velocities are chosen randomly. As stated in Section 3.2, we do not consider the physical dimensions of the ego vehicle when predicting the future density distribution. Furthermore, the goal state and the distribution for the initial state is given.

Next, we compute the reference trajectories for each environment with the three optimization methods while measuring the computation times. Two example environments and the optimized trajectories are presented in Fig. 5.6. For visualization purposes, we only consider non-moving obstacles in the displayed environments.

The collision risks and goal costs for the optimized reference trajectories are evaluated by sampling a large number of initial states from the given initial distribution and predicting the resulting state and density trajectories. With these values, we can compute the collision and goal costs as described in Section 4.1. While we use the neural density predictor for approximating the density and state trajectories in the optimization process, we will utilize the system dynamics and the Liouville equation to calculate the true trajectories in the evaluation procedure.

We will use the following criteria for the comparison:




(a) Example 1.

(b) Example 2.

Figure 5.6.: Optimized trajectories in artificial, stationary environment. Obstacles are colored in tones of gray and black - the darker the color, the higher the probability that the position is occupied by an obstacle.

- **Failure Rate**: The failure rate describes the percentage of environments where the considered method did not find a solution within five minutes. Failure cases are not considered when computing the average computation time or the cost scores.
- **Computation Time**: The average computation time for each optimization method can be computed with

$$\frac{1}{E}\sum_{j=1}^{E} T_{\square}^{(j)},$$

where $\square \in \{\text{grad}, \text{search}, \text{sampling}\}$ denotes the optimization method, $E$ is the number of evaluated environments where the method did not fail and $T_{\square}^{(j)}$ is the computation time which method $\square$ takes to find a valid solution in environment $j$.

- **Collision Risk Increase (CRI)**: We first calculate the difference of the collision risk of the considered method and the minimum reported collision risk for each environment. The CRI can then be computed by taking the average over all environments, i.e.,

$$\mathrm{CRI}_{\square} = \frac{1}{E}\sum_{j=1}^{E} \left(J^{(j)}_{\mathrm{coll},\,\square} - \min_{\triangle}\; J^{(j)}_{\mathrm{coll},\,\triangle}\right). \tag{5.13}$$

  This criteria states how much the collision risk is increased on average compared to the best possible method.

- **Goal Cost Increase (GCI)**: The GCI is computed analog to the CRI with

$$\mathrm{GCI}_{\square} = \frac{1}{E}\sum_{j=1}^{E} \left(J^{(j)}_{\mathrm{goal},\,\square} - \min_{\triangle}\; J^{(j)}_{\mathrm{goal},\,\triangle}\right). \tag{5.14}$$

- **Input Cost Increase (ICI)**: Similarly, we compute the criteria for the control effort with

$$\mathrm{ICI}_{\square} = \frac{1}{E}\sum_{j=1}^{E} \left(J^{(j)}_{\mathrm{input},\,\square} - \min_{\triangle}\; J^{(j)}_{\mathrm{input},\,\triangle}\right), \tag{5.15}$$

  where $J_{\mathrm{input},\,\square}$ is calculated with Eq. (4.3).

The results of the comparison are presented in Fig. 5.7 for 20 randomly generated environments. The detailed numerical data is given in Appendix B.3.

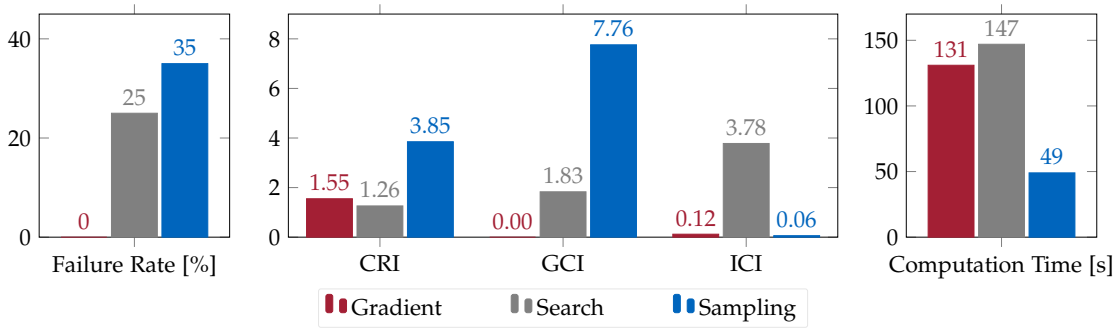



Figure 5.7.: Comparison of the optimization methods.

From the figure, we can see that the proposed gradient-based approach achieves good results in all criteria. It was always able to provide a valid solution while the search-based method failed in five and the sampling-based in seven of the twenty environments. Furthermore, the distance from the final state of the planned trajectory to the goal was on average 0.2 m and therefore much smaller compared with the solutions of the search-based (1.4 m) and the sampling-based method (2.8 m). Additionally, the

collision risk of the gradient-based approach is very small in most environments and the computation time is acceptable.

The average collision risk of the search-based method is a bit smaller compared to the gradient-based method, but the computation time is bigger and the goal and input cost are significantly higher. We could improve the goal cost by lowering the distance-to-the-goal threshold which is used for accepting trajectories, but this would significantly increase the computation times.

The sampling-based method needs the least computation time and results in input costs similar to the gradient-based approach. However, the collision and the goal cost are much larger compared to the other two optimization methods.

In general, the performance of the search- and the sampling-based methods gravely depends on the tuning of the hyperparameters, such as discretization steps, sampling parameters or acceptance thresholds, while we only need to adjust the cost function for the gradient-based approach. Especially, the sampling-based method is very sensitive - small parameter changes can enormously increase the failure rate. We did not find any hyperparameter configuration which was able to decrease the costs without resulting in a large increase in computational complexity and failure rate.

To evaluate the reliability of each method, we also have a look at the standard deviation of the computation time. The gradient-based method has by far the most stable computation time with $\sigma_{\text{grad}} = 5\,s$ since we use a fixed number of optimization iterations. The computation time of the search-based method and the sampling-based method varies significantly ($\sigma_{\text{search}} = 69\,s$ and $\sigma_{\text{sampl}} = 49\,s$).

As a consequence, we can say that the gradient-based approach is best suited for trajectory optimization. It reliably provides reference trajectories with low costs in acceptable computation times.

# 6. Motion Planning Results

In this chapter, we validate the performance of the overall motion planning approach by comparing it with baseline methods. Since most traditional motion planners cannot cope with uncertain initial states, we assume that the baseline methods start planning after the first state measurement was observed.

The proposed density planner, on the other hand, plans the reference trajectory for the given density distribution but without knowing the exact initial state in advance. After the true state was measured, solely the tracking controller gets executed to track the planned reference trajectory.

First, we present state-of-the-art motion planners and adapt them to the considered motion planning task in Section 6.1. At this point, we also show the necessity to include the collision probability in the objective function instead of constraining it to be below a threshold. In Section 6.2, we test the motion planning methods in artificially simulated environments and in environments generated from real-world data and demonstrate that the proposed approach can outperform the baseline methods.

## 6.1. Baseline Methods

In this section, we present alternative motion planning methods which will be used for validating the proposed density planner. First, we will look at conservative motion planners in Section 6.1.1 to motivate why we should consider the collision probability in the cost function. In Section 6.1.2, we will introduce a standard and a tube-based model predictive control algorithm which can be used for online motion planning. Finally, we try to approximate the optimal solution with an oracle in Section 6.1.3.

### 6.1.1. Conservative Motion Planners

In this thesis, we concentrate on motion planning in very crowded environments. As there is no guarantee that a path to the goal with zero collision probability exists, conservative motion planning algorithms which try to find a completely safe trajectory will fail. This is demonstrated by the implementation of a model predictive control (MPC) algorithm which enforces zero collision probability in its constraints. The resulting

optimization problem can be described by

$$\begin{aligned} &\min_{\mathbf{u},\mathbf{x}} J_{\text{standardMPC}}(t_h) \\ &s.t.\ \mathbf{x}(t_{k+1}) = f(\mathbf{x}(t_k), \mathbf{u}(t_k)), \quad \forall k \in \{h, h+1, ..., h+H-1\} \\ &\quad P_{\text{occ}}(\mathbf{x}(t_k), t_k) = 0, \quad \forall k \in \{h, h+1, ..., h+H\} \\ &\quad \mathbf{u}_{\min} \leq \mathbf{u}(t_k) \leq \mathbf{u}_{\max}, \quad \forall k \in \{h, h+1, ..., h+H-1\} \\ &\quad \mathbf{x}_{\min} \leq \mathbf{x}(t_k) \leq \mathbf{x}_{\max}, \quad \forall k \in \{h, h+1, ..., h+H\}, \end{aligned} \tag{6.1}$$

where $H$ is the prediction horizon, $t_h$ is the current point in time and $P_{\text{occ}}(\mathbf{x}(t_k), t_k)$ describes the probability that the position of state $\mathbf{x}(t_k)$ is occupied in the given environment at time $t_k$. Since we start planning after the initial state was observed, the density for the resulting trajectory is one and the collision probability of the trajectory, $P_{\text{coll}}(\mathbf{x}(t_k), t_k)$, is equal to $P_{\text{occ}}(\mathbf{x}(t_k), t_k)$.

Furthermore, we use a quadratic cost function which tries to minimize the control effort and the Euclidean distance from the final state to the goal:

$$\begin{aligned} J_{\text{standardMPC}}(t_h) = \alpha_{\text{input}} &\sum_{k=h}^{h+H-1} \mathbf{u}(t_k)^T \mathbf{Q}_{\text{input}} \mathbf{u}(t_k) \\ &+ \alpha_{\text{goal}} \big(\mathbf{x}(t_{h+H}) - \mathbf{x}_{\text{goal}}\big)^T \mathbf{Q}_{\text{goal}} \big(\mathbf{x}(t_{h+H}) - \mathbf{x}_{\text{goal}}\big), \end{aligned} \tag{6.2}$$

where $\alpha$ and $\mathbf{Q}$ are the same weighting factors and matrices as in the cost function of the density planner. To solve this problem at each MPC iteration, we use the nonlinear optimization framework casadi [47] and the interior point optimizer (ipopt) [48].

Next, we run the MPC in twenty environments and report the results in Appendix B.4. Although the solver is allowed to use a high number of iterations (the computation time per iteration is almost thrice as long as the admissible time for real-time control), it is not able to find a trajectory which fulfills the constraints of Problem (6.1) in 18 of 20 environments. Since the above optimization problem considers the ideal case (i.e., no disturbances or model errors), other more sophisticated motion planning approaches which consider all possible states, such as tube-based MPC [49], or the worst-case disturbances, like robust MPC [3, 50], will not be able to find a guaranteed safe path as well. By increasing the threshold of the accepted collision probability as in stochastic MPC [6, 7], the failure rate could be decreased. However, while a large threshold would lead to trajectories with high collision probability, the minimum possible value which still leads to valid trajectories depends on the uncertainty and the obstacles in the environment and cannot be predicted in advance.

Consequently, it is preferable to include the collision probability in the optimization objective as described in this thesis.

### 6.1.2. Online Motion Planners

We showed in the previous section that paths with zero collision probability cannot be found in most of the environments and that it is difficult to specify a maximum

collision probability threshold in advance. Hence, to evaluate the performance of the proposed density planner, we have to compare it with other collision-minimizing motion planning approaches. As the state-of-the-art for real-time optimal control is MPC, we will adapt two MPC algorithms to the given motion planning task by including the collision probability in the cost function.

The first MPC implementation does not consider any disturbances or model uncertainties. Thus, the optimization problem is similar to Problem (6.1) but with an extended cost function and without the collision probability constraint:

$$\begin{aligned} \min_{\mathbf{u},\mathbf{x}} \; & J_{\text{standardMPC}} + \alpha_{\text{coll}} \sum_{k=h}^{h+H} \left(P_{\text{coll}}(\mathbf{x}(t_k), t_k)\right)^2 \\ s.t. \; & \mathbf{x}(t_{k+1}) = f(\mathbf{x}(t_k), \mathbf{u}(t_k)), \quad \forall k \in \{h, h+1, ..., h+H-1\} \\ & \mathbf{u}_{\min} \leq \mathbf{u}(t_k) \leq \mathbf{u}_{\max}, \quad \forall k \in \{h, h+1, ..., h+H-1\} \\ & \mathbf{x}_{\min} \leq \mathbf{x}(t_k) \leq \mathbf{x}_{\max}, \quad \forall k \in \{h, h+1, ..., h+H\} \; . \end{aligned} \tag{6.3}$$

As the optimization problem will be solved online at each discrete point in time, the prediction horizon $H$ must be chosen sufficiently small.

In the presence of model uncertainties, the system is not able to precisely follow the nominal trajectory $\mathbf{x}(\cdot)$ which was optimized in Problem (6.3). Hence, the collision probability of the true state trajectory can be much larger than the minimized collision probability of the nominal trajectory. As a consequence, this implementation will perform badly when the state measurements are not perfect but biased.

For that reason, we implement a second motion planner which is more conservative and able to deal with uncertainties. Since the system can be kept in the neighborhood of the nominal trajectory and we want to minimize the collision probability of all possible trajectories, we choose a tube-based MPC:

First, we compute a tube which contains all possible trajectories. Then, we can minimize the overall collision probability of this tube which result in the following optimization problem:

$$\begin{aligned} \min_{\mathbf{u},\mathbf{x}} \; & J_{\text{standardMPC}} + \alpha_{\text{coll}} \sum_{k=h}^{h+H} \int_{\boldsymbol{\epsilon} \in \mathscr{T}} \left(P_{\text{coll}}(\mathbf{x}(t_k) + \boldsymbol{\epsilon}, t_k)\right)^2 d\boldsymbol{\epsilon} \\ s.t. \; & \mathbf{x}(t_{k+1}) = f(\mathbf{x}(t_k), \mathbf{u}(t_k)), \quad \forall k \in \{h, h+1, ..., h+H-1\} \\ & \mathbf{u}_{\min} \leq \mathbf{u}(t_k) \leq \mathbf{u}_{\max}, \quad \forall k \in \{h, h+1, ..., h+H-1\} \\ & \mathbf{x}_{\min} \leq \mathbf{x}(t_k) + \boldsymbol{\epsilon} \leq \mathbf{x}_{\max}, \quad \forall k \in \{h, h+1, ..., h+H\}, \; \forall \boldsymbol{\epsilon} \in \mathscr{T}, \end{aligned} \tag{6.4}$$

where the tube is defined by $\mathscr{T} = \{\boldsymbol{\epsilon} \mid ||\boldsymbol{\epsilon}|| \leq r_{\text{tube}}\}$ and the radius of the tube, $r_{\text{tube}}$, depends on the system dynamics and the uncertainties. As the collision probability only depends on the current $x$- and $y$-position, the tube will only consider deviations in these states.

Next, we want to estimate the necessary tube size. For this, we first look at the deviation of the desired position $(p_x^*(t_{k+1}), p_y^*(t_{k+1}))$ (the position which we would

reach without sensor bias) from the actual position $(p_x(t_{k+1}), p_y(t_{k+1}))$ (the position which results from the biased sensor measurement). The deviation for the $x$-position can be described by

$$\begin{aligned} p_x(t_{k+1}) - p_x^*(t_{k+1}) &= \big(p_x(t_k) + v(t_k)\cos(\theta(t_k) + \theta_{\text{bias}})\Delta t\big) - \big(p_x(t_k) + v(t_k)\cos(\theta(t_k))\Delta t\big) \\ &= v(t_k)\big(\cos(\theta(t_k) + \theta_{\text{bias}}) - \cos(\theta(t_k))\big)\Delta t. \end{aligned} \tag{6.5}$$

Since $0 \leq v \leq 10\,m/s$, $\Delta t = 0.1\,s$ and

$$\max_{\theta(t_k), |\theta_{\text{bias}}| \leq \pi/8} |\cos(\theta(t_k) + \theta_{\text{bias}}) - \cos(\theta(t_k))| = \cos(\frac{7}{16}\pi) - \cos(\frac{9}{16}\pi) \approx 0.39, \tag{6.6}$$

the bias results in a worst-case deviation of 0.39 m in $x$-direction from the nominal trajectory after one time step. Analog computations can be done for the maximum deviation in $y$-direction. As the maximum deviations in $x$- and $y$-direction cannot be reached at the same time (different heading angles $\theta(t_k)$ lead to the maxima), we can overapproximate all possible positions at the next time point by a tube with radius $r_{\text{tube}} = 0.5\,\text{m}$. Additionally, we will test a more conservative tube with $r_{\text{tube}} = 1\,\text{m}$ which contains all possible trajectories for two time steps. Larger tube sizes should result in more far-sighted behavior, but as the computational complexity of the optimization problem increases exponentially with the tube radius, we did not consider bigger tube sizes. Even with $r_{\text{tube}} = 0.5\,\text{m}$, the computational complexity is quite high such that high-performance solvers are necessary to obtain a solution in real-time. To achieve smaller computation times, we will also test a tube with $r_{\text{tube}} = 0.2\,\text{m}$.

The performance of the motion planners will be evaluated in Section 6.2.

### 6.1.3. Approximation of the Optimal Solution

In the previous section, we introduced baseline methods which can be used for online motion planning and thus, could be used as alternatives to the proposed density planner. However, these methods also have their drawbacks, but, as the minimal achievable collision probability varies from environment to environment, it is difficult to tell if a high collision risk results from bad performance of the motion planner or from a very crowded environment. Hence, it is desirable to approximate the minimal achievable costs for each environment and include them in the comparison. How we find the minimum costs will be explained in the following.

First, we assume that the minimum costs are achieved by an oracle which computes the optimal solution of Problem (6.3) with $h = 0$ and $H = N$. To be able to find the optimal solution, we will not consider computation time constraints or uncertainties. Thus, we will assume that the oracle knows the true initial state and system model exactly when starting the planning.

Problem (6.3) will then be solved by nonlinear programming. However, as the cost function is not convex due to the collision probability term, the solver is not guaranteed to find the global optimum. To overcome local minima, we will solve the optimization

problem repeatedly for different initializations, and hope that the best-found solution will be close to the optimal solution in the considered environment. However, it is important to note that the oracle approximates the minimal achievable costs but cannot be used as an alternative to the proposed density planner. Its computations are based on more information (we assume that the true initial state is given when starting the planning, while the density planner is only allowed to know the initial density distribution) and, as the oracle uses open-loop control, it is not able to deal with disturbances such as sensor bias.

The motion planning results are provided in the following section.

## 6.2. Comparison

After having introduced the baseline methods in the previous section, we now compare them with the proposed density planner. To evaluate the performance of each method, we analyze the planned trajectories which are computed as follows:

Before starting the actual motion, the density planner computes the reference trajectory for the given environment specifications, i.e., the initial density distribution, the occupation grid and the goal state are known. Next, we will sample an initial state from the density distribution and plan the input trajectories with each of the motion planners where we assume that we can measure the state every $0.1s$. The gradient-based approach will use the contraction controller to track the reference trajectory while the MPC algorithms solve Problem (6.3) or Problem (6.4) at each point in time for the horizon $H = 10$. We will test the online motion planners with perfect and with biased measurements. The oracle, on the other hand, plans the whole state trajectory after having obtained one perfect measurement of the initial state. It solves Problem (6.3) for the horizon $H = N = 100$ for 10 different random initializations. The solution with the lowest cost is chosen as the final solution of the oracle.

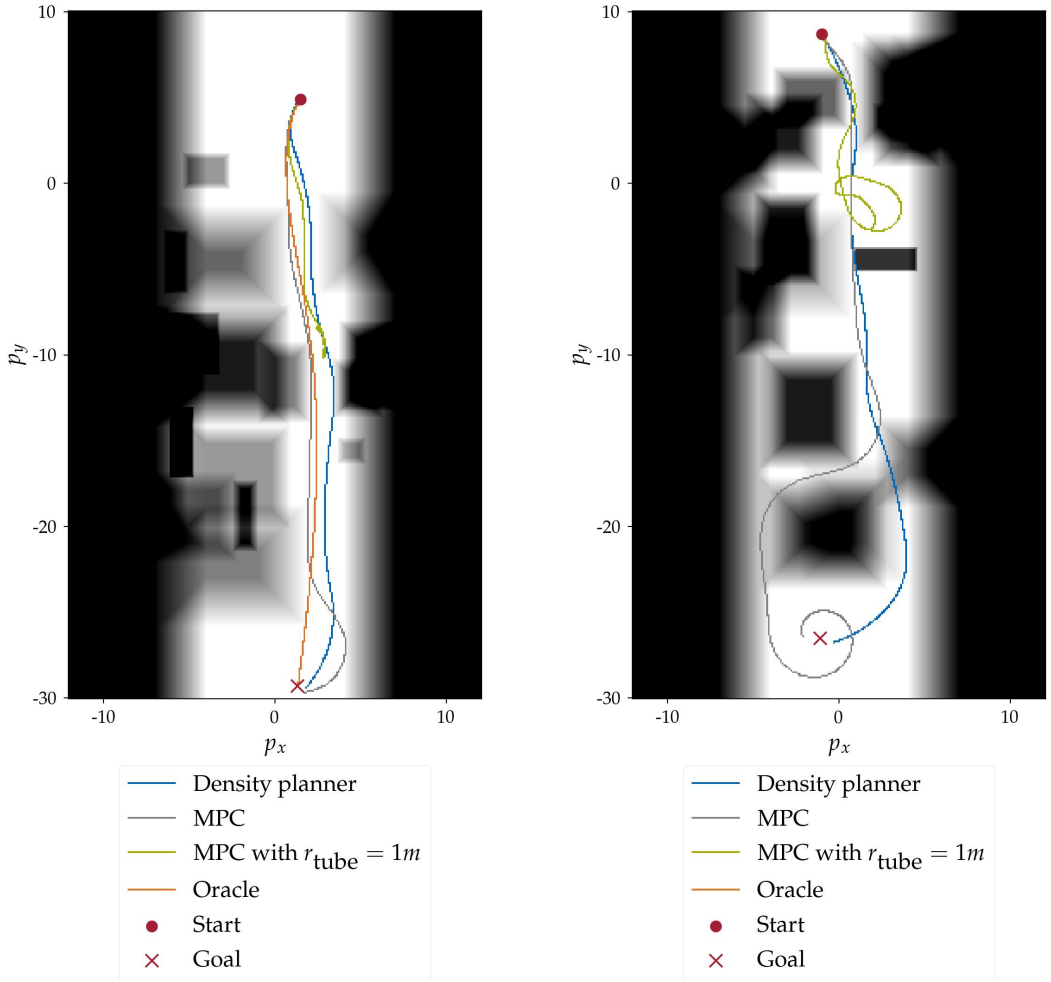
Next, the trajectories for a large number of simulated environments can be evaluated by computing the CRI, the GCI and the ICI with Eq. (5.13), Eq. (5.14) and Eq. (5.15). Furthermore, we will compute the failure rate of each planner where a failure is reported if a solution violates the state space constraints, if the final distance to the goal is bigger than 4.5m or if the resulting collision risk is higher than 10.

The results are provided in the next two subsections: In Section 6.2.1, we compare the motion planners in artificially generated environments before we show in Section 6.2.2 that the proposed motion planning concept can be easily applied to real-world data as well.

### 6.2.1. Evaluation in Artificial Environments

In this section, we analyze the motion planning methods in a large number of artificially generated environments. Analog to Section 5.3.3, the environments are created by simulating obstacles with randomly sampled positions, dimensions and uncertainties

along an artificial street. Two example environments and the trajectories planned with the density planner, the normal MPC, the tube-based MPC with $r_{\text{tube}} = 1\,\text{m}$ and the oracle are presented in Fig. 6.1. For better visualizability, we only consider stationary obstacles in the displayed environments.




(a) Example 1.

(b) Example 2.

Figure 6.1.: Final trajectories in artificial, stationary environment. Obstacles are colored in tones of gray and black - the darker the color, the higher the probability that the position is occupied by an obstacle.

In the environment illustrated in Fig. 6.1a, the density planner, the MPC and the oracle are able to find trajectories with almost zero collision probability. However, the tube-based MPC does not reach the goal since the considered tube with $r_{\text{tube}} = 1\,\text{m}$ is not able to fit through the narrow corridor at ($p_x = 2\,\text{m}$, $p_y = -11\,\text{m}$) with low collision probability.

Fig. 6.1b shows a more crowded environment which leads to higher failure rates and

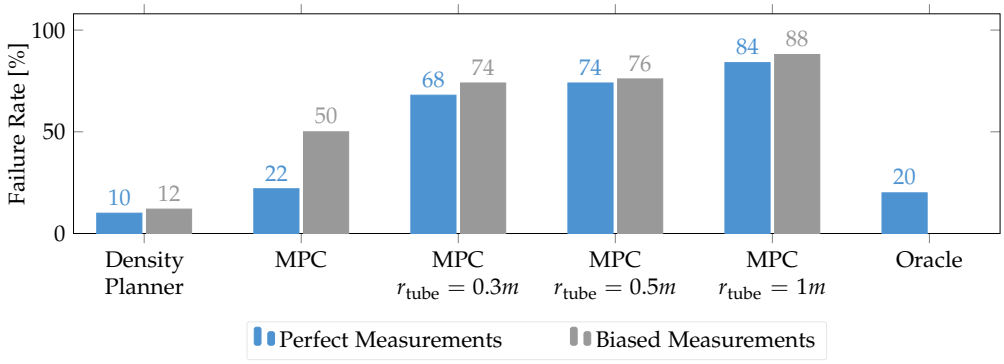
bigger costs: The oracle is not able to find a solution for optimization problem Eq. (6.3) for all initializations. Also the tube-based MPC is not able to reach the goal since the corridor between the obstacles are too narrow. The density planner finds a trajectory to the goal but collides with an obstacle at ($p_x = 1\,\mathrm{m}$, $p_y = -5\,\mathrm{m}$). The collision could be caused by the usage of inaccurate density prediction for the optimization of the reference trajectory at this position. The trajectory of the normal MPC shows low collision probability and reaches the vicinity of the goal. However, the vehicle passes the obstacle at ($p_x = 1\,\mathrm{m}$, $p_y = -5\,\mathrm{m}$) very closely. In the case of disturbances, the true trajectory would deviate from the planned path and it could come to a collision as well. Furthermore, as the MPC always minimizes the distance from the last predicted state to the goal, it only asymptotically approaches the goal position which leads to the spiral at the end of the trajectory.

Next, we evaluate the motion planners in 50 artificially generated environments. The condensed results are visualized in Fig. 6.2 and the detailed numerical data can be found in Appendix B.5.

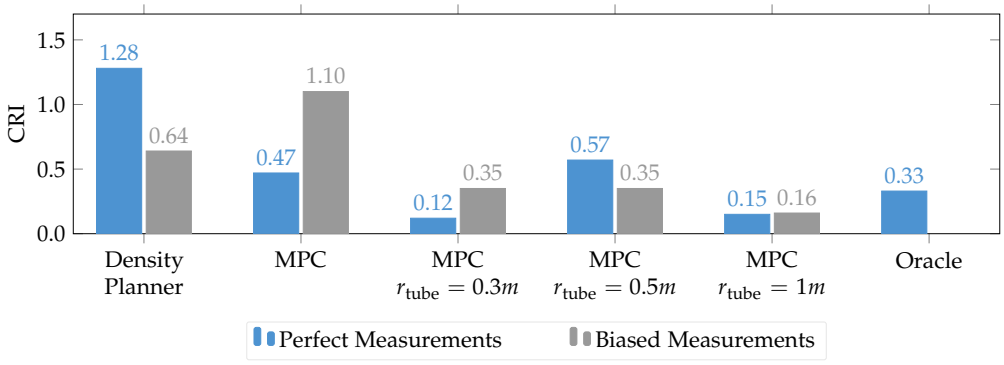
In Fig. 6.2a, we can see that the density planner is the most reliable method since it finds a valid trajectory in 45 of 50 environments when using perfect measurements. Furthermore, all trajectories end very close to the goal, with an average distance of $0.71m$, and the input cost is very low. The collision risk in Fig. 6.2b is a bit higher compared to the other motion planning approaches, but this is mainly due to the validity threshold. Even in environments where the density planner performs worse than the other methods, it is still able to reach the goal and stay below the acceptance threshold for the collision risk such that the "bad" trajectory is considered in the computations of the CRI. The other planners, on the other hand, either reach the goal with a very low collision risk or are very far off such that a failure is reported and the bad trajectory is discarded.

When the measurements are biased, the failure rate is only slightly higher which shows that the density planner is able to deal with these deterministic disturbances. Moreover, the collision risk is on average even lower compared to the case of perfect measurements. One reason for this is that the planner reports failures in the environments 37 and 41 (see Table B.9) when using biased measurements. However, when using perfect measurements the trajectories in these environments get accepted but show a much higher collision risk compared to the best possible trajectory which significantly rises the CRI.

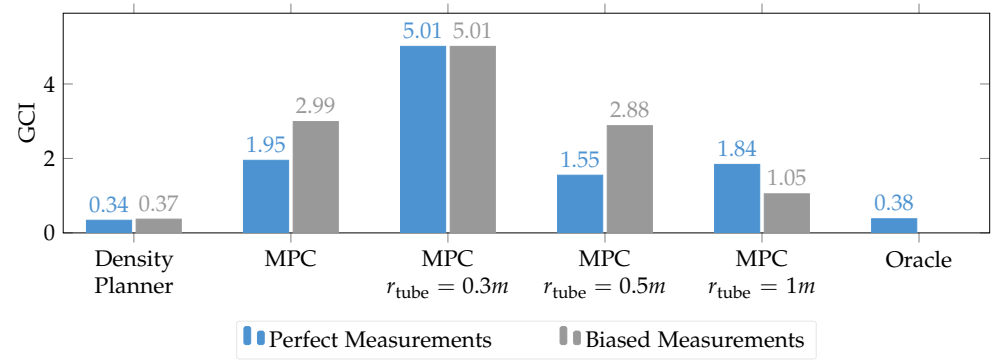
The failure rate of the normal MPC depends largely on the quality of the measurements. As it does not consider disturbances in the optimization problem, the failure rate is much higher if the sensor measurements are biased. Furthermore, the MPC has troubles to reach the goal in crowded environments: The vehicle does not move in the direction of the goal when the collision risk on the way to the goal is higher when the actual cost decrease for getting close to the goal. Since the tube-based MPC plans the trajectories to be more conservative and more safe, it has even more difficulties in crowded environments. The larger the tube, the bigger is the increase of the collision risk when driving through the crowded environment and hence, the higher are the number



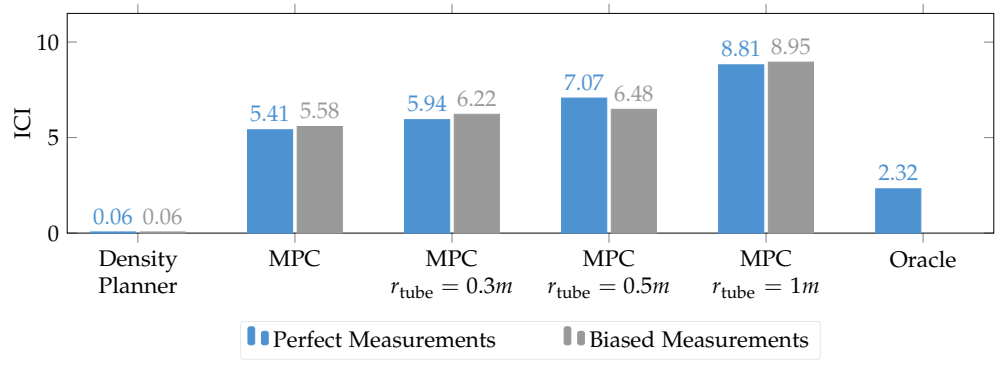

(a) Failure rate.




(b) Collision risk.




(c) Goal cost.




(d) Input cost.

Figure 6.2.: Comparison of the motion planning approaches in artificial environments.

of failures. As the tube-based MPC only reaches the goal if it has found a path with low collision probability for the whole tube, the collision risks of the accepted trajectories are very small. Additionally, we can see that the tube-based MPC is much more robust against disturbances than the normal MPC. The failure rate and the average collision risk when using biased measurements are almost equal to the ideal case when using perfect measurements. However, because of the short planning horizon, an MPC-controlled vehicle does not move very far-sighted which leads to big input changes. Consequently, the input costs in Fig. 6.2d are quite high for all the MPC algorithms.

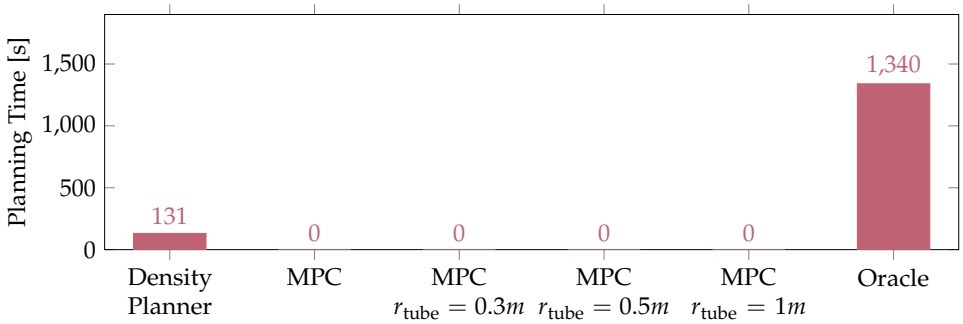
As the oracle solves the same optimization problem for the whole planning horizon at once, the resulting motion is very far-sighted which leads to small input costs. Furthermore, the collision and the goal costs are very small which indicates that the trajectories planned by the oracle are good approximations of the optimal solutions. Also the failure rate is low, but this mainly comes from the large number of differently-initialized trials. The solver is only able to find a solution for one third of the initializations on average.

Lastly, we will look at the computation times in Fig. 6.3. Fig. 6.3a shows the planning time which the algorithms need before starting the online control. The density planner takes on average 131 s to compute the reference trajectory. As the oracle solves optimization problem Eq. (6.3) for ten different initializations, the resulting planning time is very big. In addition, the oracle requires the true initial state when starting the planning while the density planner only needs the initial density distribution. The MPCs solve the corresponding optimization problems online and consequently do not need any planning time in advance. On the other hand, the online computational complexity of the MPC algorithms is very high, see Fig. 6.3b. Only the normal MPC and the tube-based MPC with $r_{\text{tube}} = 0.2\,\text{m}$ are able to solve the optimization problem in approximately real-time (we use a time step of $\Delta t = 0.1\,\text{s}$) with the utilized hardware[1]. The density planner only needs to compute the output of the neural contraction controller to follow the precomputed reference trajectory. Hence, the resulting online computational complexity is very low. As the oracle plans the whole motion in advance, it does not perform any online calculations.

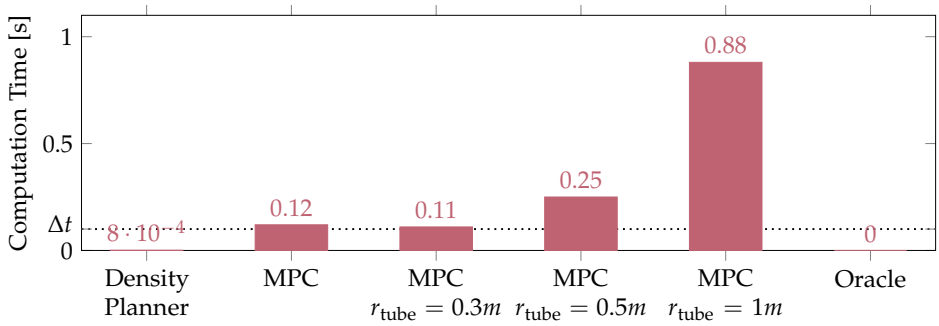
### 6.2.2. Validation with Real-World Data

After having analyzed the motion planners in the artificially generated environments, we now want to show that they can be applied to real-world environments without modifications. The only requirement is that we know the occupancy maps at each point in time. Normally, these maps would be forecasted with environment predictors by using current and past observations. Here, we will use the inD dataset [51] which contains a collection of naturalistic vehicle, bicyclist and pedestrian trajectories recorded at German intersections by drones. First, we transform the recordings to occupation maps and, as environment predictions are usually very uncertain, we add some more uncertainty around each traffic participant. After having specified a start and a goal

[1]CPU: AMD Ryzen Threadripper 3990X (2.9GHz), GPU: NVIDIA A4000 16GB




(a) Offline planning time.




(b) Online computation time.

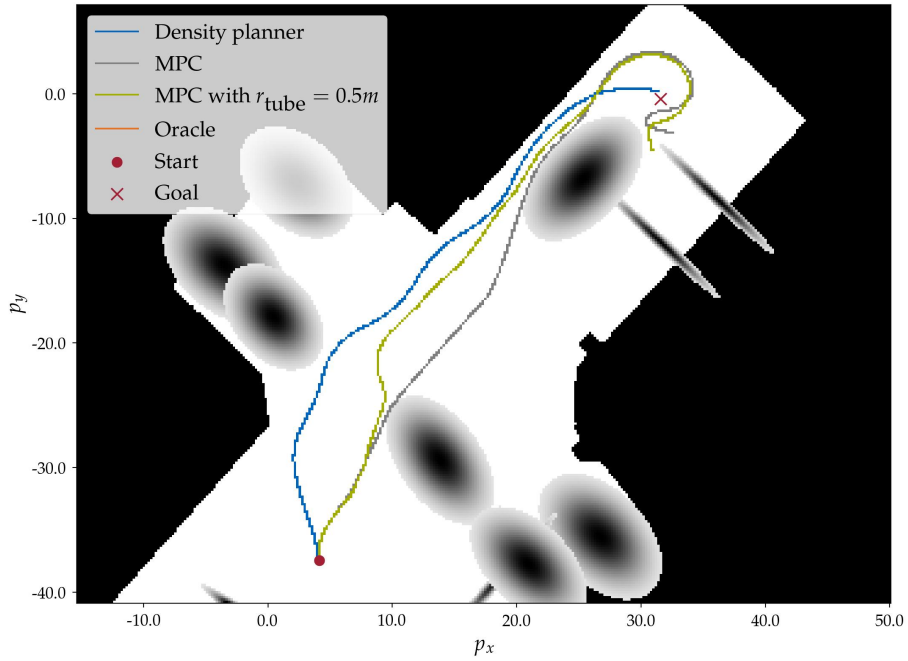
Figure 6.3.: Computation times of the motion planning approaches.

state, the motion planners can be applied in the same way as for the artificially generated environments.

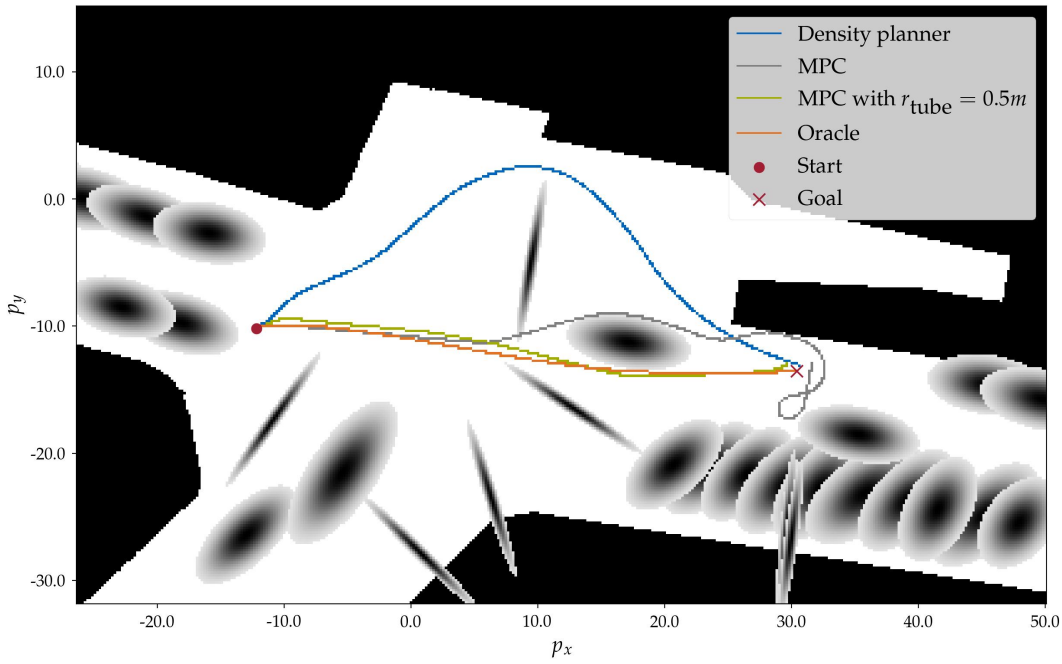
To evaluate the performance, we will look at the recordings at three different intersections. The resulting occupation maps at a random point in time as well as the trajectories planned by the density planner, by the normal MPC, by a tube-based MPC with $r_{\text{tube}} = 0.5\,\text{m}$ and by the oracle are displayed in Fig. 6.4. We choose the tube radius to be 0.5 m as a compromise between robustness and low computational complexity.

In Fig. 6.4a, all three online planners are able to reach the goal with zero collision probability. While the trajectory of the normal MPC passes very close to the obstacles, the density planner and the tube-based MPC always keep some distance to the obstacles and hence, their trajectories are more robust against disturbances. The oracle does not find a solution in this environment. In general, we observed that the oracle often fails in stationary environments while it has quite a low failure rate in dynamic environments. The reason for this could be that moving obstacles lead to a time-varying cost landscape and less stationary local optima where the solver can get stuck.

In Fig. 6.4b, all motion planners succeed in finding a trajectory to the goal with low collision risk. Almost the same applies to Fig. 6.4c, only the trajectory of the oracle leads to high collision risks. It seems to be stuck in a local optimum since small changes of the trajectory, especially at ($p_x = 25\,\text{m}$, $p_y = -15\,\text{m}$), would lead to higher collision probabilities.

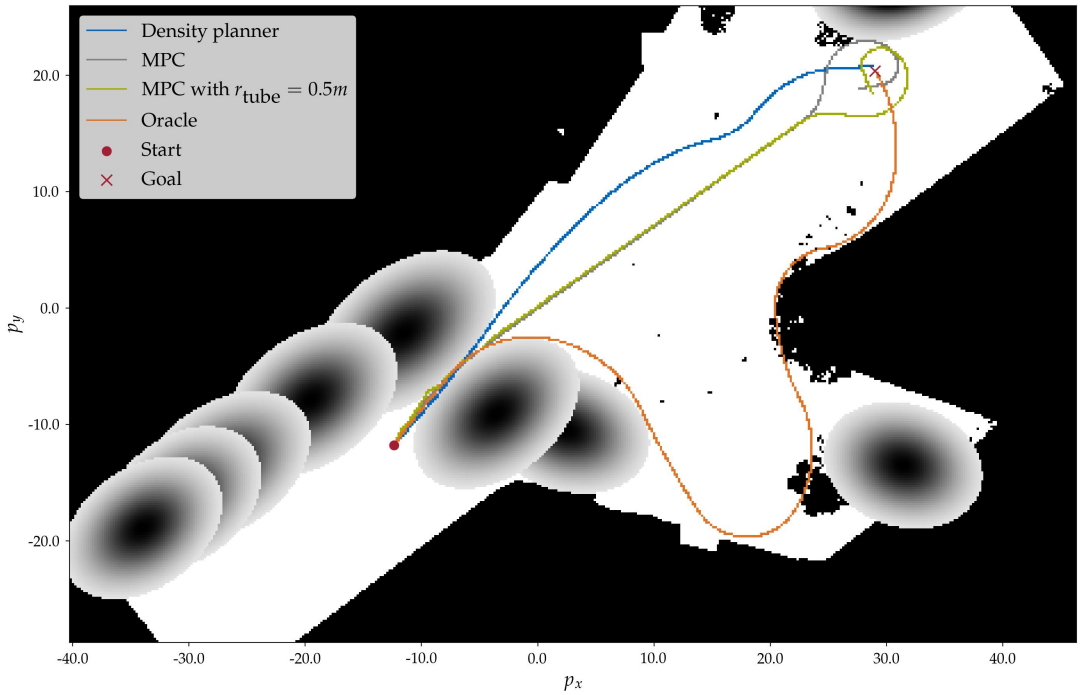



(a) Intersection 1.




(b) Intersection 2.

Figure 6.4.: Final trajectories in stationary environment generated with real-world data. Other traffic participants are visualized by ellipses in tones of gray (the darker the color, the higher the probability that the position is occupied). The area beyond the street boundaries or outside of the observation area of the drones is colored in black.

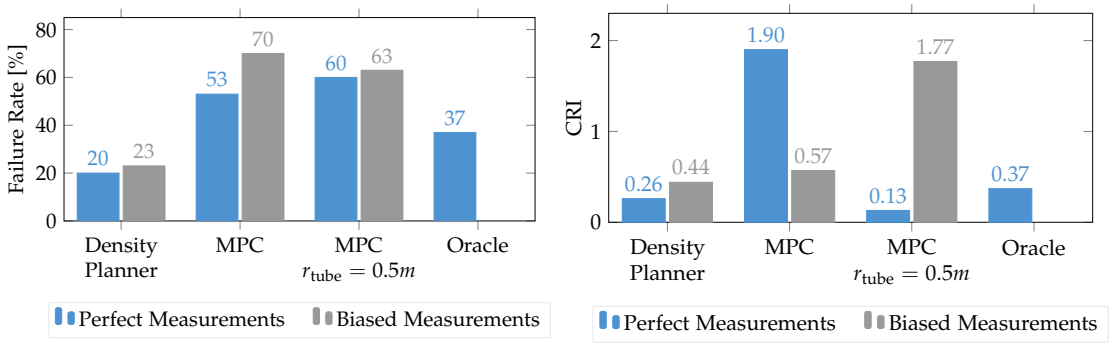



(c) Intersection 3.

Figure 6.4.: Final trajectories in stationary environment generated with real-world data. Other traffic participants are visualized by ellipses in tones of gray (the darker the color, the higher the probability that the position is occupied). The area beyond the street boundaries or outside of the observation area of the drones is colored in black.

Next, we will look at the results from applying the density planner, the normal MPC, the tube-based MPC with $r_{\text{tube}} = 0.5\,\text{m}$ and the oracle to the real-world environments. To generate the data, we randomly sample start and goal positions for ten random time periods in recordings of each intersection, and compute the costs for each planner. The condensed results are visualized in Fig. 6.5 and the numerical data can be found in Appendix B.6.

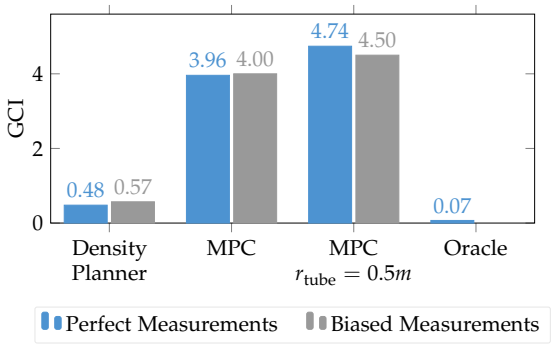
The most expressive criteria is the failure rate in Fig. 6.5a since it shows us in how many environments the motion planners find a trajectory leading to the goal with acceptable collision risk. The best results are achieved by the density planner; it finds a path to the goal in 24 of 30 environments for the case of perfect measurements. When using biased measurements, the failure rate is only slightly higher which again underlines the robustness of the planner. Also the failure rate of the oracle is acceptable; it finds valid trajectories in 19 environments. The normal MPC fails in more than half of the environments with a big difference between the ideal and the biased case. As the tube-based MPC is more robust, the failure rate between both measurement types is



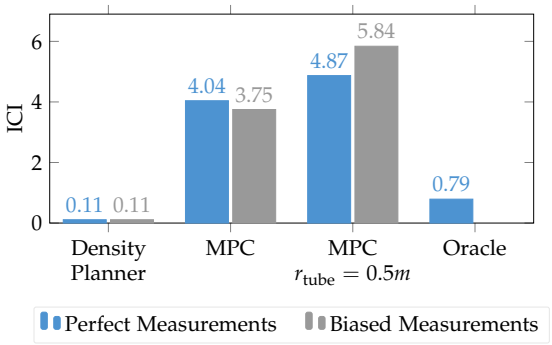

(a) Failure rate.




(b) Collision risk.




(c) Goal cost.




(d) Input cost.

Figure 6.5.: Comparison of the motion planning approaches in real-world environments.

very small. If we compare the performance of both MPCs, we can say that the normal MPC shows better results in the ideal case since it is less conservative but, if we have disturbances such as sensor bias, the tube-based MPC should be used.

In Fig. 6.5b, the CRI is illustrated. We can see that the density planner shows very good results which are even better than the oracle when using ideal measurements. The collision risk is only slightly higher when using biased measurements.

As the collision risks of the MPCs vary a lot (see Table B.16 to Table B.18), it is difficult to differentiate between a failure case and bad performance. Hence, the CRI is less meaningful for these methods. For instance, the difference in the CRI between the biased and ideal case for the normal MPC, comes mainly from two environments, seed 7 in Table B.17 and seed 0 in Table B.18. In these environments, the collision risk of the MPC in the ideal case is just below the acceptance threshold but much worse than the best possible collision risk. As a consequence, these environments considerably influence the CRI. However, in the case of biased measurements, the collision risk is above the acceptance threshold such that a failure is reported and the environments do not get considered in the computation of the CRI. Similar things can be observed for the tube-based MPC (see seed 1 in Table B.17 and seed 8 in Table B.18).

Fig. 6.5c and Fig. 6.5c display the GCI and ICI for each planner. The density planner and the oracle have very small goal and input costs, while the costs for the MPC

algorithms are quite high which mostly results from the short prediction horizon.

Except for the big variations of the MPC collision risks, the results are about the same as for the artificially generated environments. The density planner is again the most reliable motion planning method. Its trajectories lead to the goal while having a low collision probability. Hence, the proposed approach can successfully be used for motion planning in the real world.

# 7. Conclusions

This chapter concludes the thesis. First, we want to describe the limitations of the proposed motion planning concept in Section 7.1. In Section 7.2, future research directions will be presented.

## 7.1. Limitations

The weakest point of the proposed motion planner is the neural density predictor. Although the predictions are in general quite good, in some cases the density network is not able to approximate the true distribution, and by optimizing the reference trajectory with bad density predictions, the resulting state trajectories can lead to high collision costs.

Furthermore, to use the motion planner in the real world, we need good predictions for the evolution of the environment. With the utilized hardware, the planner takes about 130s to plan the reference trajectory which means that the environment predictor must be able to look far (130s plus the planning horizon) ahead. Predictions with such a long prediction horizon can usually only be generated if we assume that an exact map of the traffic infrastructure is known and that the ego vehicle can communicate with possible obstacles about their future positions. However, the planning time can be considerably decreased by using more powerful hardware and a more efficient implementation in C or C++.

The cost function also leaves room for improvements. By summing the cost over all sample trajectories, regions in the state space with high density have an unproportional high influence on the overall cost compared to regions with smaller density: First, there are more trajectories in this area which get considered in the cost function, and second, the cost of each trajectory gets weighted by its density. Additionally, the structure of the collision cost makes it difficult for the ego vehicle to be close to obstacles whose positions are known exactly since the resulting high occupation probability gradients lead to a large repulsion.

Another limitation of the approach is the exchangeability of the system dynamics. Firstly, we have to train a new contraction controller and a new density predictor when the system dynamics are changed. The training times largely depend on the dimensionality of the system and are usually quite long. Furthermore, the usage of the LE for the supervised learning of the density network requires that the system dynamics are deterministic and known exactly. Additionally, to be able to use the contraction controller, the dynamics must be linear in the inputs. However, the motion

planning concept does not require the usage of contraction theory. In general, any well-performing tracking controller could be used as long as the density network is able to predict the resulting closed-loop dynamics. The contraction controller has the advantage that bounds for the maximum tracking error can be provided. Furthermore, the resulting collision risks should be lower since all state trajectories are guaranteed to stay close to the reference trajectories, even in the case of disturbances.

## 7.2. Future Work

The usage of other tracking controllers and their influence on the accuracy of the density predictor should be investigated in the future. We can imagine that simpler controller concepts lead to better density predictions since the approximation of the resulting closed-loop dynamics is easier.

On the other handy, more sophisticated network types and training methods could be used to improve the density network, e.g., the use of recurrent neural network seems promising and dropout layers or skip connections could be tested.

The current network approximates the density distribution pointwise, i.e., it predicts the density of a large number of input samples, and with interpolation and normalization, the final distribution can be estimated. Another approach which should be considered in the future is the usage of multi-dimensional convolutional neural networks to approximate the whole density distribution at once. This way, we could get a differentiable expression of the collision probability which consequently could be minimized directly instead of minimizing the collision risk of density-weighted sample trajectories.

Another possible future research direction would be the derivation of safety certificates. If we were able to bound the maximum density prediction errors, we could provide performance guarantees for the motion planning concept by analyzing the optimization method. These guarantees could also be probabilistic.

As most disturbances are of stochastic nature, we would like to generalize the motion planning approach to stochastic systems in the future. For this, it would be necessary to solve the FPE instead of the LE to generate the training data. However, the FPE is difficult to solve directly, especially for high-dimensional systems, and we would need a large amount of data for the supervised learning of the density network. As a consequence, the usage of a physics-informed neural network [52] seems promising. Instead of relying purely on training data, the physics-informed neural network could enforce the FPE directly on its outputs by including the equation in its loss function.

Furthermore, by using a simpler tracking controller (without the rigorous convergence guarantees of contraction theory), the approach could be applied to nonlinear system which are not affine in the control input.

Finally, we would like to apply the motion planning approach to real vehicles and evaluate its performance while using occupation maps generated by true environment predictors. To predict the true density distribution and collision probability, the vehicle

dimensions must be considered. Additionally, to decrease the prediction horizon for the environment predictors and thus make the predictions more reliable, the planning time should be significantly reduced. This could be done by using more powerful computing hardware, optimizing the implementation for speed and translating the code to a faster programming language such as C or C++.

# A. Implementation Details

## A.1. State and Input Space

Table A.1.: State and Input Space for the Neural Contraction Controller

| | **Original State and Input Space from [43]** | **Extended State and Input Space for Motion Planning** |
|---|---|---|
| **x-position** $p_x$ [m] | $p_x \in [-5, 5]$ | $p_x \in [-50, 50]$ |
| **y-position** $p_y$ [m] | $p_y \in [-5, 5]$ | $p_y \in [-50, 50]$ |
| **Heading angle** $\theta$ [rad] | $\theta \in [-\pi, \pi]$ | $\theta \in [-\pi, 3\pi]$ |
| **Velocity** $v$ [m/s] | $v \in [1, 2]$ | $v \in [0, 10]$ |
| **Initial deviation of the reference trajectory** $\mathbf{x}_e(0) = \mathbf{x}(0) - \mathbf{x}_*(0)$ | $\mathbf{x}_e^{(i)}(0) \geq [-1, -1, -1, -1]^T$, $\mathbf{x}_e^{(i)}(0) \leq [1, 1, 1, 1]^T$ | $\mathbf{x}_e^{(i)}(0) \geq [-2, -2, -1, -1]^T$, $\mathbf{x}_e^{(i)}(0) \leq [2, 2, 1, 1]^T$ |
| **Angular velocity** $\omega$ [rad/s] | $\omega \in [-3, 3]$ | $\omega \in [-3, 3]$ |
| **Longitudinal acceleration** $a$ [m/$s^2$] | $a \in [-3, 3]$ | $a \in [-3, 3]$ |

## A.2. Parameters

Table A.2.: Cost Function of Gradient-based Optimization Algorithm

| **Parameter** | **Description** | **Value** |
|---|---|---|
| $\alpha_{\text{bounds}}$ | weight for keeping the trajectory in the valid state space | 10 |
| $\alpha_{\text{collision}}$ | weight for collision cost | 0.1 |
| $\alpha_{\text{goal}}$ | weight for goal cost | 0.01 |
| $\alpha_{\text{input}}$ | weight for input cost | 1e-4 |

# B. Numerical Results

## B.1. Neural Contraction Controller

Table B.1.: Computation Time [ms]

| **Measurement** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | **Mean** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **NN** | 0.76 | 0.76 | 0.77 | 0.77 | 0.76 | 0.81 | 0.77 | 0.76 | 0.78 | 0.77 | 0.771 |

## B.2. Neural Density Predictor

Table B.2.: Computation Time [s]

| **Measurement** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | **Mean** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **NN Prediction** | 0.37 | 0.27 | 0.28 | 0.29 | 0.30 | 0.28 | 0.33 | 0.28 | 0.32 | 0.30 | 0.302 |
| **LE and Dynamics** | 10.68 | 10.70 | 10.95 | 11.21 | 10.99 | 11.86 | 12.44 | 12.06 | 13.8 | 11.53 | 11.622 |

## B.3. Evaluation of the Optimization Methods

Table B.3.: Collision Costs

| Environment Seed | Optimization Method | | |
|---|---|---|---|
| | Gradient-based | Search-based | Sampling-based |
| 0 | 0.062 | 1.346 | 1.632 |
| 1 | 2.617 | 0.939 | 4.606 |
| 2 | 3.862 | - | 6.221 |
| 3 | 11.989 | 1.597 | 8.013 |
| 4 | 0.047 | 2.200 | 2.556 |
| 5 | 6.470 | - | - |
| 6 | 0.420 | 1.833 | - |
| 7 | 0.315 | - | - |
| 8 | 0.633 | 1.208 | - |
| 9 | 0.643 | 1.102 | 0.858 |
| 10 | 0.069 | 0.261 | 13.948 |
| 11 | 0.096 | 0.663 | 1.373 |
| 12 | 0.313 | 1.598 | 7.841 |
| 13 | 0.541 | 10.379 | - |
| 14 | 12.955 | 0.911 | 2.798 |
| 15 | 9.959 | 5.783 | 7.692 |
| 16 | 6.182 | - | - |
| 17 | 0.964 | - | 4.044 |
| 18 | 3.046 | 0.403 | 4.098 |
| 19 | 0.355 | 1.532 | - |

Table B.4.: Goal Costs

| Environment Seed | Optimization Method | | |
|---|---|---|---|
| | Gradient-based | Search-based | Sampling-based |
| 0 | 0.087 | 3.112 | 12.962 |
| 1 | 0.218 | 9.793 | 8.035 |
| 2 | 0.038 | - | 10.801 |
| 3 | 0.048 | 2.423 | 2.474 |
| 4 | 0.104 | 2.135 | 3.000 |
| 5 | 0.410 | - | - |
| 6 | 0.135 | 0.641 | - |
| 7 | 0.392 | - | - |
| 8 | 0.228 | 0.587 | - |
| 9 | 0.115 | 1.111 | 10.761 |
| 10 | 0.018 | 1.074 | 6.178 |
| 11 | 0.196 | 2.441 | 9.829 |
| 12 | 0.116 | 0.610 | 7.727 |
| 13 | 0.050 | 1.250 | - |
| 14 | 0.090 | 1.877 | 0.870 |
| 15 | 0.107 | 0.912 | 16.809 |
| 16 | 2.750 | - | - |
| 17 | 0.697 | - | 5.006 |
| 18 | 0.101 | 0.696 | 8.306 |
| 19 | 0.077 | 0.533 | - |

Table B.5.: Control Costs

| Environment Seed | Optimization Method | | |
|---|---|---|---|
| | Gradient-based | Search-based | Sampling-based |
| 0 | 0.377 | 4.600 | 0.497 |
| 1 | 0.551 | 4.500 | 0.257 |
| 2 | 0.515 | - | 0.647 |
| 3 | 1.320 | 3.600 | 0.324 |
| 4 | 0.477 | 4.800 | 0.740 |
| 5 | 0.981 | - | - |
| 6 | 0.328 | 4.600 | - |
| 7 | 0.933 | - | - |
| 8 | 0.546 | 2.700 | - |
| 9 | 0.363 | 2.500 | 0.590 |
| 10 | 0.510 | 5.400 | 0.427 |
| 11 | 0.526 | 4.100 | 0.457 |
| 12 | 0.452 | 4.300 | 0.478 |
| 13 | 0.489 | 3.200 | - |
| 14 | 0.448 | 3.400 | 0.282 |
| 15 | 0.816 | 3.800 | 0.429 |
| 16 | 0.562 | - | - |
| 17 | 0.635 | - | 0.385 |
| 18 | 0.705 | 4.600 | 0.467 |
| 19 | 0.356 | 4.300 | - |

Table B.6.: Computation Time

| Environment Seed | Optimization Method | | |
|---|---|---|---|
| | Gradient-based | Search-based | Sampling-based |
| 0 | 131.241 | 219.305 | 27.638 |
| 1 | 128.880 | 213.447 | 21.743 |
| 2 | 138.425 | - | 15.631 |
| 3 | 147.981 | 93.570 | 12.651 |
| 4 | 124.687 | 92.860 | 139.947 |
| 5 | 129.103 | - | - |
| 6 | 124.629 | 96.795 | - |
| 7 | 126.193 | - | - |
| 8 | 127.950 | 132.996 | - |
| 9 | 126.676 | 167.600 | 45.570 |
| 10 | 130.001 | 67.073 | 50.723 |
| 11 | 127.741 | 245.814 | 24.265 |
| 12 | 131.843 | 89.575 | 16.705 |
| 13 | 130.173 | 74.287 | - |
| 14 | 129.357 | 181.503 | 36.685 |
| 15 | 130.098 | 216.660 | 63.228 |
| 16 | 132.400 | - | - |
| 17 | 130.531 | - | 14.419 |
| 18 | 132.716 | 245.415 | 167.713 |
| 19 | 131.034 | 59.336 | - |

## B.4. Safe MPC

Table B.7.: Motion Planning Results for the Safe MPC

| **Environment Seed** | Goal Cost | Input Cost | Computation Time for One MPC Iteration [s] |
|---|---|---|---|
| 0 | - | - | - |
| 1 | - | - | - |
| 2 | - | - | - |
| 3 | - | - | - |
| 4 | 6.291 | 7.621 | 0.28 |
| 5 | - | - | - |
| 6 | - | - | - |
| 7 | - | - | - |
| 8 | - | - | - |
| 9 | - | - | - |
| 10 | 3.344 | 7.370 | 0.30 |
| 11 | - | - | - |
| 12 | - | - | - |
| 13 | - | - | - |
| 14 | - | - | - |
| 15 | - | - | - |
| 16 | - | - | - |
| 17 | - | - | - |
| 18 | - | - | - |
| 19 | - | - | - |

## B.5. Evaluation of the Motion Planning Methods in Artificial Environment

Table B.8.: Collision Costs in Artificial Environment, Part 1

| Environment Seed | Motion Planning Approach | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Density Planner | | MPC | | MPC $r_{\text{tube}} = 0.3m$ | | MPC $r_{\text{tube}} = 0.5m$ | | MPC $r_{\text{tube}} = 1m$ | | Oracle |
| | ideal | bias | ideal | bias | ideal | bias | ideal | bias | ideal | bias | ideal |
| 0 | 0.000 | 0.302 | 0.070 | 0.165 | 0.000 | - | 0.235 | 0.070 | - | 1.193 | 0.000 |
| 1 | 3.118 | 0.079 | 0.056 | 0.155 | 0.032 | 0.083 | 1.722 | 0.007 | 0.000 | 0.000 | 0.000 |
| 2 | 2.848 | 0.173 | 2.660 | 0.765 | 0.076 | 0.004 | - | 0.517 | - | - | 1.713 |
| 3 | 4.210 | 0.797 | 2.382 | 2.625 | - | 0.000 | - | - | 0.143 | 0.111 | - |
| 4 | 0.067 | 0.144 | 0.028 | - | 0.007 | 0.124 | 0.000 | 0.185 | - | - | 0.000 |
| 5 | 6.633 | 4.119 | - | 1.804 | 0.024 | - | 1.496 | 3.896 | - | - | 0.719 |
| 6 | 0.320 | 0.134 | 0.199 | - | 0.025 | 0.000 | 0.087 | - | - | - | 0.002 |
| 7 | 0.340 | 0.300 | 0.468 | 1.300 | 0.000 | 0.000 | 0.262 | 16.852 | 0.001 | 0.000 | 7.683 |
| 8 | 0.655 | 0.917 | 0.175 | - | 0.005 | 0.381 | - | - | - | - | 0.000 |
| 9 | 0.568 | 1.030 | 1.591 | 2.293 | 0.001 | 0.030 | 0.181 | - | - | 0.361 | 0.001 |
| 10 | 0.018 | 0.106 | 0.061 | 0.073 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.002 | 0.000 |
| 11 | 0.094 | 0.009 | 1.969 | 2.102 | 0.000 | 0.211 | 0.125 | 0.122 | - | - | 0.000 |
| 12 | 0.381 | 0.183 | 0.183 | - | 0.017 | 0.005 | 0.000 | 0.000 | 0.080 | - | 0.000 |
| 13 | 0.509 | 0.733 | 0.145 | 0.868 | - | - | 0.046 | 0.465 | 0.685 | 0.841 | 0.161 |
| 14 | 8.568 | 0.417 | 0.856 | - | 0.021 | 0.035 | 0.428 | 0.152 | - | - | 0.064 |
| 15 | - | 0.345 | 0.129 | - | 0.170 | 1.752 | 11.481 | - | - | - | 0.033 |
| 16 | 4.733 | 3.775 | - | - | 0.359 | 0.005 | 0.030 | 0.361 | 0.187 | 0.202 | - |
| 17 | 0.659 | 2.568 | 0.943 | 1.286 | 0.025 | 0.567 | 1.057 | 0.580 | - | - | - |
| 18 | 3.462 | 0.062 | 0.189 | 0.208 | 0.176 | 0.009 | 0.035 | 0.025 | 0.466 | 0.162 | - |
| 19 | 0.361 | 1.040 | 2.040 | 4.327 | 0.011 | 0.003 | 4.208 | - | - | - | 0.000 |
| 20 | 0.862 | 0.897 | 0.239 | 0.350 | 0.469 | 0.313 | 0.215 | 0.784 | 0.238 | 0.344 | 0.764 |
| 21 | 2.193 | 0.216 | 2.277 | 1.008 | 0.000 | 0.000 | 0.000 | 0.854 | 0.000 | 0.000 | 0.135 |
| 22 | - | - | - | - | 0.053 | - | 0.070 | - | - | - | 2.889 |
| 23 | 2.377 | 0.033 | 0.148 | - | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 24 | 0.398 | 0.956 | 0.079 | - | 0.000 | 0.014 | 0.000 | 0.102 | 0.000 | 0.000 | 0.109 |

Table B.9.: Collision Costs in Artificial Environment, Part 2

| Environment Seed | Motion Planning Approach | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Density Planner | | MPC | | MPC $r_{\text{tube}} = 0.3m$ | | MPC $r_{\text{tube}} = 0.5m$ | | MPC $r_{\text{tube}} = 1m$ | | Oracle |
| | ideal | bias | ideal | bias | ideal | bias | ideal | bias | ideal | bias | ideal |
| 25 | - | - | 0.373 | - | 0.956 | - | 0.633 | 0.627 | 0.845 | - | 1.991 |
| 26 | - | - | - | - | 0.000 | 0.000 | 0.000 | 0.000 | - | 0.036 | 0.139 |
| 27 | 1.541 | 1.199 | 0.184 | 0.161 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | - | - |
| 28 | 0.087 | 0.258 | 0.090 | - | 0.000 | 0.109 | 0.000 | 0.000 | 0.000 | 0.000 | 0.027 |
| 29 | 0.000 | 0.000 | - | - | 0.000 | 0.045 | 0.159 | 0.152 | 0.169 | 0.171 | 0.000 |
| 30 | 0.169 | 0.440 | - | 0.512 | 1.119 | 0.016 | 0.000 | 0.175 | 0.000 | 0.000 | 0.000 |
| 31 | 0.174 | 0.481 | 0.129 | 0.329 | 34.409 | 2.242 | 2.596 | 0.196 | 0.397 | 5.272 | 0.290 |
| 32 | 0.170 | 0.137 | 0.156 | - | 0.003 | 0.243 | 0.000 | 0.000 | 0.000 | - | 0.012 |
| 33 | - | - | - | - | 1.083 | 0.386 | 0.608 | 0.204 | - | 0.063 | 0.086 |
| 34 | 2.000 | 1.879 | 0.049 | - | 0.327 | - | 0.417 | - | 0.000 | 0.000 | - |
| 35 | 0.484 | 0.489 | 0.823 | 1.367 | 0.000 | 0.013 | 0.000 | 0.000 | 0.000 | 0.000 | - |
| 36 | 1.297 | 1.653 | 1.813 | 7.945 | - | - | - | - | - | - | 0.118 |
| 37 | 3.517 | - | - | - | 0.005 | 0.000 | 0.000 | 3.839 | - | 0.000 | - |
| 38 | 0.119 | 0.049 | 0.130 | - | 0.000 | 0.005 | 0.637 | - | 0.000 | - | 0.002 |
| 39 | 0.013 | 0.009 | 0.128 | 0.236 | 0.000 | - | 0.000 | 0.000 | 0.000 | 0.036 | 0.000 |
| 40 | 0.110 | 0.062 | 0.473 | - | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.158 |
| 41 | 4.097 | 19.367 | 0.015 | 0.249 | 0.000 | - | 0.649 | 0.408 | - | 0.012 | 0.302 |
| 42 | 1.227 | 0.573 | 0.068 | 0.150 | 0.000 | - | 0.000 | 0.000 | 0.000 | - | 0.043 |
| 43 | 0.251 | 0.021 | 1.045 | - | 0.020 | 0.045 | 0.000 | - | 0.000 | 0.000 | 0.000 |
| 44 | 0.327 | 0.323 | 57.018 | - | 0.217 | 0.390 | 0.005 | 0.065 | - | 0.007 | 0.471 |
| 45 | 5.948 | 6.338 | 2.874 | 3.277 | 3.287 | - | 1.150 | 1.098 | - | 2.424 | - |
| 46 | 0.064 | 0.100 | 0.297 | 0.351 | 0.000 | 0.000 | - | 0.418 | 0.000 | 0.085 | - |
| 47 | 0.116 | 0.321 | 0.048 | - | 0.015 | 0.014 | 0.202 | 0.091 | 0.150 | 0.193 | 0.065 |
| 48 | 0.169 | 0.103 | 0.098 | 0.102 | 14.236 | 0.044 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 49 | 2.814 | 1.580 | 0.132 | 33.674 | 14.596 | 12.560 | - | 0.148 | - | 0.003 | 0.001 |

Table B.10.: Goal Costs in Artificial Environment, Part 1

| Environment Seed | Motion Planning Approach | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Density Planner** | | **MPC** | | **MPC** $r_{\text{tube}} = 0.3m$ | | **MPC** $r_{\text{tube}} = 0.5m$ | | **MPC** $r_{\text{tube}} = 1m$ | | **Oracle** |
| | ideal | bias | ideal | bias | ideal | bias | ideal | bias | ideal | bias | ideal |
| 0 | 0.275 | 1.293 | 5.247 | 11.691 | 0.388 | - | 51.230 | 64.301 | - | 103.315 | 0.000 |
| 1 | 0.165 | 0.015 | 0.882 | 0.882 | 528.004 | 527.573 | 0.976 | 7.368 | 527.620 | 531.308 | 0.000 |
| 2 | 0.254 | 0.019 | 0.914 | 0.934 | 722.738 | 728.893 | - | 134.857 | - | - | 0.011 |
| 3 | 0.031 | 0.006 | 6.016 | 2.382 | - | 536.019 | - | - | 0.880 | 0.951 | - |
| 4 | 0.270 | 0.866 | 5.519 | - | 13.473 | 2.673 | 29.034 | 14.124 | - | - | 0.000 |
| 5 | 0.059 | 0.071 | - | 18.873 | 799.220 | - | 2.377 | 1.854 | - | - | 5.000 |
| 6 | 0.157 | 0.007 | 3.683 | - | 4.929 | 870.207 | 4.132 | - | - | - | 0.000 |
| 7 | 0.412 | 1.046 | 0.712 | 0.453 | 734.031 | 729.418 | 186.722 | 135.265 | 146.635 | 143.608 | 0.009 |
| 8 | 0.114 | 0.641 | 0.939 | - | 0.929 | 14.483 | - | - | - | - | 0.000 |
| 9 | 0.302 | 0.828 | 7.530 | 2.951 | 4.126 | 19.350 | 133.762 | - | - | 50.031 | 0.000 |
| 10 | 0.211 | 0.734 | 1.777 | 2.019 | 3.231 | 3.498 | 0.965 | 0.887 | 0.538 | 0.770 | 0.000 |
| 11 | 0.291 | 0.007 | 0.896 | 0.963 | 0.846 | 0.937 | 0.750 | 5.085 | - | - | 0.000 |
| 12 | 0.336 | 0.129 | 0.892 | - | 0.944 | 0.986 | 0.660 | 0.909 | 0.907 | - | 0.000 |
| 13 | 0.332 | 0.038 | 0.891 | 17.905 | - | - | 0.885 | 0.994 | 0.894 | 0.963 | 0.000 |
| 14 | 0.119 | 0.017 | 6.327 | - | 504.502 | 504.785 | 132.027 | 505.395 | - | - | 0.000 |
| 15 | - | 0.055 | 0.938 | - | 0.872 | 0.835 | 139.896 | - | - | - | 0.000 |
| 16 | 5.407 | 2.773 | - | - | 12.818 | 37.894 | 125.779 | 23.471 | 149.988 | 115.013 | - |
| 17 | 1.868 | 3.986 | 0.961 | 0.998 | 710.313 | 0.739 | 4.581 | 43.530 | - | - | - |
| 18 | 0.495 | 0.078 | 0.948 | 0.992 | 812.868 | 241.252 | 0.900 | 0.985 | 4.678 | 0.988 | - |
| 19 | 0.279 | 1.586 | 0.982 | 0.935 | 19.010 | 19.313 | 2.270 | - | - | - | 0.000 |
| 20 | 0.161 | 0.171 | 0.342 | 0.782 | 194.582 | 135.116 | 205.800 | 181.627 | 216.865 | 500.619 | 0.000 |
| 21 | 0.301 | 0.034 | 0.627 | 0.908 | 654.711 | 654.819 | 372.833 | 465.114 | 389.985 | 526.118 | 0.000 |
| 22 | - | - | - | - | 115.780 | - | 150.726 | - | - | - | 15.746 |
| 23 | 0.294 | 0.067 | 0.950 | - | 0.847 | 0.951 | 241.487 | 0.664 | 268.291 | 251.977 | 0.000 |
| 24 | 0.226 | 0.712 | 0.912 | - | 65.326 | 0.642 | 69.341 | 69.285 | 77.592 | 75.921 | 0.097 |

Table B.11.: Goal Costs in Artificial Environment, Part 2

| Environment Seed | Motion Planning Approach | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Density Planner | | MPC | | MPC $r_{\text{tube}} = 0.3m$ | | MPC $r_{\text{tube}} = 0.5m$ | | MPC $r_{\text{tube}} = 1m$ | | Oracle |
| | ideal | bias | ideal | bias | ideal | bias | ideal | bias | ideal | bias | ideal |
| 25 | - | - | 0.999 | - | 0.899 | - | 128.423 | 0.728 | 121.509 | - | 0.000 |
| 26 | - | - | - | - | 424.531 | 408.041 | 425.830 | 419.849 | - | 439.783 | 0.000 |
| 27 | 0.518 | 0.624 | 0.665 | 0.493 | 888.700 | 882.288 | 886.679 | 891.183 | 643.756 | - | - |
| 28 | 0.271 | 0.383 | 0.884 | - | 666.133 | 665.211 | 678.258 | 677.579 | 697.742 | 712.626 | 0.000 |
| 29 | 0.268 | 0.145 | - | - | 273.896 | 180.039 | 163.926 | 238.887 | 295.935 | 288.194 | 0.000 |
| 30 | 0.321 | 0.932 | - | 7.512 | 18.072 | 0.864 | 790.486 | 55.090 | 219.189 | 62.914 | 0.000 |
| 31 | 0.093 | 0.239 | 0.951 | 0.734 | 123.076 | 0.647 | 155.850 | 22.917 | 156.617 | 165.642 | 0.000 |
| 32 | 0.184 | 0.014 | 0.950 | - | 49.774 | 26.228 | 0.957 | 685.219 | 663.717 | - | 0.026 |
| 33 | - | - | - | - | 466.365 | 426.256 | 942.770 | 664.859 | - | 778.761 | 0.000 |
| 34 | 1.228 | 0.950 | 0.885 | - | 118.365 | - | 146.346 | - | 809.737 | 465.872 | - |
| 35 | 1.833 | 1.841 | 0.795 | 0.779 | 122.367 | 151.106 | 128.105 | 127.512 | 70.153 | 138.993 | - |
| 36 | 0.279 | 0.962 | 0.690 | 0.969 | - | - | - | - | - | - | 0.000 |
| 37 | 0.431 | - | - | - | 896.961 | 891.848 | 891.430 | 616.927 | - | 931.708 | - |
| 38 | 0.296 | 0.004 | 0.561 | - | 57.104 | 71.660 | 148.541 | - | 4.420 | - | 0.000 |
| 39 | 0.135 | 0.103 | 0.981 | 0.969 | 0.930 | - | 0.984 | 0.854 | 1.694 | 1.810 | 0.000 |
| 40 | 0.245 | 0.037 | 17.763 | - | 442.778 | 443.437 | 453.453 | 453.576 | 475.854 | 469.342 | 10.352 |
| 41 | 1.914 | 2.800 | 0.998 | 0.586 | 0.633 | - | 92.052 | 640.321 | - | 511.318 | 0.009 |
| 42 | 0.092 | 0.017 | 0.889 | 0.841 | 562.547 | - | 579.725 | 571.651 | 604.502 | - | 0.000 |
| 43 | 0.197 | 0.012 | 50.559 | - | 29.046 | 46.808 | 590.285 | - | 615.575 | 611.712 | 0.000 |
| 44 | 0.293 | 0.001 | 49.772 | - | 42.795 | 134.105 | 139.383 | 142.453 | - | 225.353 | 0.050 |
| 45 | 1.190 | 1.352 | 0.999 | 0.944 | 182.752 | - | 476.945 | 478.133 | - | 210.234 | - |
| 46 | 0.148 | 0.225 | 29.065 | 27.965 | 855.903 | 851.089 | - | 307.933 | 471.849 | 473.105 | - |
| 47 | 0.279 | 0.044 | 0.924 | - | 317.179 | 315.992 | 324.941 | 324.729 | 478.964 | 479.922 | 0.000 |
| 48 | 0.249 | 0.164 | 0.196 | 0.346 | 0.797 | 20.687 | 0.653 | 0.304 | 0.759 | 0.890 | 0.000 |
| 49 | 0.029 | 0.062 | 0.981 | 0.951 | 3.451 | 59.480 | - | 349.035 | - | 390.130 | 0.000 |

Table B.12.: Input Costs in Artificial Environment, Part 1

| Environment Seed | Motion Planning Approach | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Density Planner | | MPC | | MPC $r_{\text{tube}} = 0.3m$ | | MPC $r_{\text{tube}} = 0.5m$ | | MPC $r_{\text{tube}} = 1m$ | | Oracle |
| | ideal | bias | ideal | bias | ideal | bias | ideal | bias | ideal | bias | ideal |
| 0 | 0.377 | 0.377 | 6.704 | 6.807 | 3.928 | - | 11.545 | 9.687 | - | 11.767 | 0.250 |
| 1 | 0.551 | 0.551 | 6.087 | 6.326 | 4.373 | 8.923 | 7.266 | 7.842 | 6.501 | 7.987 | 0.483 |
| 2 | 0.515 | 0.515 | 7.502 | 7.240 | 5.769 | 5.183 | - | 11.949 | - | - | 9.322 |
| 3 | 1.320 | 1.320 | 6.222 | 6.773 | - | 5.176 | - | - | 8.500 | 10.140 | - |
| 4 | 0.477 | 0.477 | 6.383 | - | 8.810 | 9.067 | 10.193 | 8.595 | - | - | 2.853 |
| 5 | 0.981 | 0.981 | - | 6.847 | 6.817 | - | 6.832 | 7.368 | - | - | 5.485 |
| 6 | 0.328 | 0.328 | 6.146 | - | 7.784 | 1.570 | 7.932 | - | - | - | 2.326 |
| 7 | 0.933 | 0.933 | 4.590 | 4.608 | 4.504 | 4.014 | 6.938 | 11.503 | 10.958 | 10.894 | 5.752 |
| 8 | 0.546 | 0.546 | 6.091 | - | 5.756 | 8.921 | - | - | - | - | 0.159 |
| 9 | 0.363 | 0.363 | 8.258 | 8.240 | 4.838 | 5.814 | 11.614 | - | - | 14.133 | 2.326 |
| 10 | 0.510 | 0.510 | 6.257 | 6.194 | 7.645 | 6.469 | 6.009 | 5.271 | 8.459 | 8.392 | 0.011 |
| 11 | 0.526 | 0.526 | 6.039 | 6.328 | 8.545 | 7.210 | 5.568 | 7.537 | - | - | 0.301 |
| 12 | 0.452 | 0.452 | 6.060 | - | 5.622 | 7.786 | 3.850 | 7.908 | 11.851 | - | 0.244 |
| 13 | 0.489 | 0.489 | 5.663 | 6.830 | - | - | 9.589 | 10.305 | 9.426 | 10.207 | 3.769 |
| 14 | 0.448 | 0.448 | 6.829 | - | 7.933 | 7.243 | 8.312 | 8.480 | - | - | 4.840 |
| 15 | - | 0.816 | 6.876 | - | 6.056 | 6.260 | 11.443 | - | - | - | 5.015 |
| 16 | 0.562 | 0.562 | - | - | 9.750 | 9.086 | 8.270 | 11.765 | 8.365 | 10.389 | - |
| 17 | 0.635 | 0.635 | 6.055 | 6.302 | 7.918 | 6.918 | 9.260 | 10.699 | - | - | - |
| 18 | 0.705 | 0.705 | 6.408 | 6.556 | 6.141 | 7.242 | 7.198 | 6.824 | 11.967 | 11.616 | - |
| 19 | 0.356 | 0.356 | 6.751 | 6.953 | 7.857 | 6.821 | 11.125 | - | - | - | 1.395 |
| 20 | 0.553 | 0.553 | 6.478 | 6.491 | 10.501 | 8.559 | 9.749 | 8.574 | 11.329 | 9.603 | 3.980 |
| 21 | 1.181 | 1.181 | 7.921 | 7.493 | 4.308 | 4.444 | 5.598 | 10.659 | 7.122 | 5.327 | 5.487 |
| 22 | - | - | - | - | 7.690 | - | 10.971 | - | - | - | 3.080 |
| 23 | 0.516 | 0.516 | 6.451 | - | 5.767 | 4.133 | 5.663 | 7.296 | 7.513 | 11.145 | 0.141 |
| 24 | 0.656 | 0.656 | 6.857 | - | 2.773 | 3.926 | 3.064 | 7.029 | 2.011 | 2.351 | 4.563 |

Table B.13.: Input Costs in Artificial Environment, Part 2

| Environment Seed | Motion Planning Approach | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Density Planner | | MPC | | MPC $r_{\text{tube}} = 0.3m$ | | MPC $r_{\text{tube}} = 0.5m$ | | MPC $r_{\text{tube}} = 1m$ | | Oracle |
| | ideal | bias | ideal | bias | ideal | bias | ideal | bias | ideal | bias | ideal |
| 25 | - | - | 6.291 | - | 7.949 | - | 9.844 | 3.535 | 11.567 | - | 5.014 |
| 26 | - | - | - | - | 2.898 | 2.659 | 5.512 | 5.285 | - | 9.163 | 4.061 |
| 27 | 0.399 | 0.399 | 3.771 | 3.509 | 3.235 | 2.241 | 2.054 | 1.862 | 9.133 | - | - |
| 28 | 0.756 | 0.756 | 6.023 | - | 3.866 | 4.613 | 3.887 | 4.220 | 4.886 | 3.982 | 2.350 |
| 29 | 0.487 | 0.487 | - | - | 6.125 | 8.310 | 8.866 | 6.430 | 9.602 | 9.462 | 2.106 |
| 30 | 0.661 | 0.661 | - | 7.581 | 7.690 | 7.220 | 6.738 | 7.708 | 8.713 | 6.915 | 2.787 |
| 31 | 0.930 | 0.930 | 4.743 | 3.833 | 10.072 | 6.055 | 12.084 | 10.039 | 14.431 | 12.749 | 2.716 |
| 32 | 0.544 | 0.544 | 6.520 | - | 6.154 | 6.839 | 5.389 | 4.379 | 5.060 | - | 3.958 |
| 33 | - | - | - | - | 9.560 | 8.084 | 10.738 | 9.056 | - | 10.981 | 2.291 |
| 34 | 1.016 | 1.016 | 4.774 | - | 9.211 | - | 9.789 | - | 8.856 | 10.030 | - |
| 35 | 0.621 | 0.621 | 6.215 | 6.144 | 2.092 | 5.795 | 2.221 | 2.078 | 2.545 | 3.207 | - |
| 36 | 0.350 | 0.350 | 7.458 | 7.253 | - | - | - | - | - | - | 4.319 |
| 37 | 0.331 | - | - | - | 4.948 | 5.154 | 4.333 | 8.431 | - | 2.702 | - |
| 38 | 0.687 | 0.687 | 4.218 | - | 6.304 | 5.873 | 7.622 | - | 8.530 | - | 3.749 |
| 39 | 0.478 | 0.478 | 5.432 | 6.785 | 4.699 | - | 9.854 | 6.524 | 8.453 | 9.220 | 0.084 |
| 40 | 0.886 | 0.886 | 6.152 | - | 1.906 | 2.814 | 2.427 | 2.525 | 3.938 | 4.429 | 5.915 |
| 41 | 1.632 | 1.632 | 1.741 | 3.614 | 2.265 | - | 10.013 | 10.425 | - | 11.158 | 6.767 |
| 42 | 0.886 | 0.886 | 9.190 | 8.948 | 4.257 | - | 4.202 | 4.338 | 5.720 | - | 2.437 |
| 43 | 0.711 | 0.711 | 5.855 | - | 7.957 | 6.723 | 3.814 | - | 4.062 | 5.408 | 3.279 |
| 44 | 1.236 | 1.236 | 6.430 | - | 8.753 | 7.113 | 8.926 | 7.537 | - | 11.508 | 5.937 |
| 45 | 2.335 | 2.335 | 8.707 | 5.251 | 7.535 | - | 9.283 | 7.848 | - | 10.021 | - |
| 46 | 1.114 | 1.114 | 5.348 | 5.331 | 2.021 | 2.023 | - | 5.518 | 5.711 | 6.843 | - |
| 47 | 1.094 | 1.094 | 6.009 | - | 4.086 | 3.865 | 7.544 | 6.818 | 9.847 | 9.264 | 4.201 |
| 48 | 0.682 | 0.682 | 3.879 | 3.791 | 8.267 | 7.935 | 7.558 | 6.504 | 7.118 | 7.038 | 0.316 |
| 49 | 0.736 | 0.736 | 5.480 | 4.759 | 9.589 | 9.494 | - | 7.542 | - | 9.331 | 3.481 |

Table B.14.: Computation Times (MPC: average computation time per iteration, Oracle: overall computation time), Part 1

| Environment Seed | Motion Planning Approach | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | MPC | | MPC $r_{\text{tube}} = 0.3m$ | | MPC $r_{\text{tube}} = 0.5m$ | | MPC $r_{\text{tube}} = 1m$ | | Oracle |
| | ideal | bias | ideal | bias | ideal | bias | ideal | bias | ideal |
| 0 | 0.06 | 0.07 | 0.12 | - | 0.26 | 0.25 | - | 0.88 | 1005.91 |
| 1 | 0.06 | 0.06 | 0.11 | 0.11 | 0.26 | 0.26 | 0.87 | 0.87 | 1263.46 |
| 2 | 0.09 | 0.09 | 0.11 | 0.11 | - | 0.25 | - | - | 1830.58 |
| 3 | 0.11 | 0.11 | - | 0.11 | - | - | 0.90 | 0.88 | - |
| 4 | 0.06 | - | 0.11 | 0.11 | 0.26 | 0.25 | - | - | 1701.15 |
| 5 | - | 0.10 | 0.11 | - | 0.25 | 0.25 | - | - | 1540.91 |
| 6 | 0.09 | - | 0.11 | 0.11 | 0.25 | - | - | - | 1373.18 |
| 7 | 0.09 | 0.08 | 0.11 | 0.11 | 0.25 | 0.25 | 0.89 | 0.89 | 1747.57 |
| 8 | 0.08 | - | 0.12 | 0.11 | - | - | - | - | 1028.28 |
| 9 | 0.10 | 0.10 | 0.11 | 0.11 | 0.25 | - | - | 0.87 | 1183.48 |
| 10 | 0.06 | 0.07 | 0.11 | 0.11 | 0.26 | 0.26 | 0.90 | 0.91 | 722.04 |
| 11 | 0.08 | 0.09 | 0.12 | 0.12 | 0.27 | 0.26 | - | - | 947.12 |
| 12 | 0.08 | - | 0.12 | 0.12 | 0.26 | 0.26 | 0.90 | - | 442.60 |
| 13 | 0.11 | 0.10 | - | - | 0.26 | 0.26 | 0.90 | 0.90 | 1423.77 |
| 14 | 0.09 | - | 0.11 | 0.11 | 0.25 | 0.25 | - | - | 1778.37 |
| 15 | - | 0.07 | - | 0.12 | 0.12 | 0.26 | - | - | 1403.82 |
| 16 | - | - | 0.11 | 0.11 | 0.25 | 0.25 | 0.87 | 0.88 | - |
| 17 | 0.13 | 0.13 | 0.11 | 0.12 | 0.26 | 0.25 | - | - | - |
| 18 | 0.10 | 0.09 | 0.11 | 0.12 | 0.26 | 0.26 | 0.88 | 0.91 | - |
| 19 | 0.10 | 0.11 | 0.12 | 0.12 | 0.25 | - | - | - | 1471.70 |
| 20 | 0.11 | 0.12 | 0.11 | 0.11 | 0.25 | 0.25 | 0.86 | 0.86 | 1390.21 |
| 21 | 0.12 | 0.12 | 0.11 | 0.11 | 0.25 | 0.25 | 0.87 | 0.88 | 1479.86 |
| 22 | - | - | - | 0.11 | - | 0.26 | - | - | 1662.03 |
| 23 | 0.10 | - | 0.12 | 0.12 | 0.25 | 0.26 | 0.88 | 0.87 | 1026.21 |
| 24 | 0.12 | - | 0.11 | 0.12 | 0.25 | 0.25 | 0.88 | 0.87 | 1376.66 |

Table B.15.: Computation Times (MPC: average computation per iteration, Oracle: overall computation time), Part 2

| Environment Seed | Motion Planning Approach | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | MPC | | MPC $r_{\text{tube}} = 0.3m$ | | MPC $r_{\text{tube}} = 0.5m$ | | MPC $r_{\text{tube}} = 1m$ | | Oracle |
| | ideal | bias | ideal | bias | ideal | bias | ideal | bias | ideal |
| 25 | 0.12 | - | 0.12 | - | 0.25 | 0.26 | 0.87 | - | 783.60 |
| 26 | - | - | 0.12 | 0.11 | 0.25 | 0.25 | - | 0.88 | 1240.37 |
| 27 | 0.16 | 0.16 | 0.11 | 0.11 | 0.25 | 0.25 | 0.87 | - | - |
| 28 | 0.13 | - | 0.11 | 0.11 | 0.25 | 0.25 | 0.87 | 0.87 | 1258.37 |
| 29 | - | - | 0.11 | 0.11 | 0.25 | 0.26 | 0.87 | 0.88 | 872.85 |
| 30 | - | 0.14 | 0.11 | 0.12 | 0.26 | 0.26 | 0.87 | 0.87 | 856.90 |
| 31 | 0.15 | 0.15 | 0.12 | 0.12 | 0.25 | 0.26 | 0.87 | 0.87 | 717.44 |
| 32 | 0.14 | - | 0.11 | 0.12 | 0.27 | 0.27 | 1.14 | - | 1231.44 |
| 33 | - | - | 0.11 | 0.11 | 0.25 | 0.25 | - | 0.87 | 1313.27 |
| 34 | 0.14 | - | 0.11 | - | 0.25 | - | 0.87 | 0.87 | - |
| 35 | 0.15 | 0.14 | 0.11 | 0.11 | 0.25 | 0.25 | 0.88 | 0.87 | - |
| 36 | 0.16 | 0.16 | - | - | - | - | - | - | 1322.80 |
| 37 | - | - | 0.11 | 0.11 | 0.25 | 0.25 | - | 0.86 | - |
| 38 | 0.12 | - | 0.11 | 0.11 | 0.25 | - | 0.87 | - | 1489.50 |
| 39 | 0.13 | 0.13 | 0.12 | - | 0.26 | 0.26 | 0.87 | 0.87 | 244.90 |
| 40 | 0.12 | - | 0.11 | 0.11 | 0.25 | 0.25 | 0.88 | 0.88 | 1561.83 |
| 41 | 0.12 | 0.12 | 0.12 | - | 0.25 | 0.25 | - | 0.86 | 1329.98 |
| 42 | 0.13 | 0.13 | 0.11 | - | 0.25 | 0.25 | 0.86 | - | 1578.71 |
| 43 | 0.14 | - | 0.11 | 0.11 | 0.25 | - | 0.88 | 0.88 | 189.31 |
| 44 | 0.15 | - | 0.11 | 0.11 | 0.25 | 0.25 | - | 0.88 | 1694.98 |
| 45 | 0.15 | 0.16 | 0.11 | - | 0.26 | 0.26 | - | 0.88 | - |
| 46 | 0.12 | 0.12 | 0.11 | 0.11 | - | 0.25 | 0.87 | 0.87 | - |
| 47 | 0.12 | - | 0.11 | 0.11 | 0.25 | 0.25 | 0.87 | 0.87 | 1781.09 |
| 48 | 0.13 | 0.13 | 0.12 | 0.11 | 0.26 | 0.26 | 0.90 | 0.90 | 376.97 |
| 49 | 0.12 | 0.12 | 0.11 | 0.12 | - | 0.25 | - | 0.88 | 836.87 |

## B.6. Evaluation of the Motion Planning Methods in Real-world Environment

Table B.16.: Collision Costs in Real-world Environment, Intersection 1 (Recording 8)

| Environment Seed | Motion Planning Approach | | | | | | |
|---|---|---|---|---|---|---|---|
| | Density Planner | | MPC | | MPC $r_{\text{tube}} = 0.5m$ | | Oracle |
| | ideal | bias | ideal | bias | ideal | bias | ideal |
| 0 | 0.000 | 0.351 | 0.000 | 0.255 | - | - | 0.000 |
| 1 | 0.000 | 0.000 | 1.095 | - | 0.000 | - | 0.000 |
| 2 | 0.963 | 4.674 | 16.413 | 789.681 | 1.098 | - | - |
| 3 | 291.159 | 94.469 | 1.073 | - | - | - | 191.920 |
| 4 | 380.574 | 865.553 | 12.855 | 14.790 | - | - | - |
| 5 | 0.000 | 0.016 | 105.995 | - | - | 0.489 | 0.000 |
| 6 | 0.064 | 0.210 | 17.600 | - | - | 51.906 | - |
| 7 | 0.317 | 1.383 | 0.286 | 381.818 | 0.063 | 0.000 | 0.000 |
| 8 | 0.105 | 0.000 | 383.276 | 190.654 | 0.000 | 0.000 | 0.000 |
| 9 | 0.000 | 0.000 | 76.216 | 32.889 | 0.637 | 0.000 | 0.000 |

Table B.17.: Collision Costs in Real-world Environment, Intersection 2 (Recording 26)

| Environment Seed | Motion Planning Approach | | | | | | |
|---|---|---|---|---|---|---|---|
| | Density Planner | | MPC | | MPC $r_{\text{tube}} = 0.5m$ | | Oracle |
| | ideal | bias | ideal | bias | ideal | bias | ideal |
| 0 | 0.000 | 0.000 | 0.396 | 0.553 | 0.000 | 5.039 | 871.701 |
| 1 | 0.000 | 0.000 | 14.011 | 15.972 | 28.062 | 9.551 | 17.388 |
| 2 | 0.000 | 0.000 | 7.752 | 1.568 | 0.000 | 0.853 | 0.000 |
| 3 | 684.435 | 199.938 | 52.485 | 18.194 | 0.000 | 0.000 | 0.060 |
| 4 | 243.478 | 924.664 | 45.859 | 101.700 | - | 68.362 | - |
| 5 | 0.000 | 0.000 | 0.835 | 1.418 | 31.414 | 37.408 | - |
| 6 | 0.000 | 0.000 | 12.828 | 1.662 | 14.434 | 11.509 | 5.587 |
| 7 | 2.333 | 2.271 | 6.568 | 302.201 | 2.152 | 32.060 | 2.198 |
| 8 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 2.646 | 0.000 |
| 9 | 685.123 | - | 7.685 | 7.205 | 7.877 | 7.498 | 6.766 |

Table B.18.: Collision Costs in Real-world Environment, Intersection 3 (Recording 30)

| Environment Seed | Motion Planning Approach | | | | | | |
|---|---|---|---|---|---|---|---|
| | Density Planner | | MPC | | MPC $r_{\text{tube}} = 0.5m$ | | Oracle |
| | ideal | bias | ideal | bias | ideal | bias | ideal |
| 0 | 0.000 | 380.180 | 8.693 | 292.344 | 0.091 | 0.377 | 0.000 |
| 1 | 0.638 | 0.567 | 0.538 | 0.642 | 0.611 | 0.329 | 1.726 |
| 2 | 1.418 | 0.530 | 0.649 | 2.865 | 0.610 | 0.587 | 0.448 |
| 3 | 1.751 | 0.022 | 16.219 | 26.759 | 0.136 | 0.000 | 191.607 |
| 4 | 1.417 | 1.557 | 0.692 | 0.685 | 0.140 | 0.014 | 0.000 |
| 5 | 0.737 | 0.699 | 4.872 | 0.746 | 2.721 | 1.202 | 0.815 |
| 6 | 1.544 | 1.293 | 2.625 | 3.661 | 0.224 | - | 0.196 |
| 7 | 0.280 | 1.681 | 979.343 | - | 0.186 | 0.176 | - |
| 8 | 0.000 | 0.000 | 3.275 | 14.059 | 249.939 | 9.040 | 0.000 |
| 9 | 62.224 | 24.321 | 3.174 | 194.731 | 10.594 | - | - |

Table B.19.: Goal Costs in Real-world Environment, Intersection 1 (Recording 8)

| Environment Seed | Motion Planning Approach | | | | | | |
|---|---|---|---|---|---|---|---|
| | Density Planner | | MPC | | MPC $r_{\text{tube}} = 0.5m$ | | Oracle |
| | ideal | bias | ideal | bias | ideal | bias | ideal |
| 20 | 0.066 | 0.329 | 10.106 | 10.013 | - | - | 0.000 |
| 21 | 0.208 | 0.008 | 20.684 | - | 14.677 | - | 0.000 |
| 22 | 0.415 | 0.047 | 0.784 | 0.826 | 54.322 | - | - |
| 23 | 3.514 | 3.153 | 39.070 | - | - | - | 0.000 |
| 24 | 0.851 | 1.831 | 5.385 | 3.949 | - | - | - |
| 25 | 0.125 | 0.011 | 11.391 | - | - | 0.985 | 0.000 |
| 26 | 0.185 | 0.582 | 0.803 | - | - | 0.408 | - |
| 27 | 0.262 | 0.871 | 0.796 | 0.875 | 14.863 | 77.812 | 0.000 |
| 28 | 2.155 | 3.035 | 0.926 | 0.886 | 0.731 | 43.096 | 0.000 |
| 29 | 0.292 | 1.095 | 6.690 | 10.389 | 20.593 | 19.478 | 0.000 |

Table B.20.: Goal Costs in Real-world Environment, Intersection 2 (Recording 26)

| Environment Seed | Motion Planning Approach | | | | | | |
|---|---|---|---|---|---|---|---|
| | Density Planner | | MPC | | MPC $r_{\text{tube}} = 0.5m$ | | Oracle |
| | ideal | bias | ideal | bias | ideal | bias | ideal |
| 0 | 0.167 | 0.529 | 0.535 | 4.491 | 0.753 | 0.932 | 0.000 |
| 1 | 0.344 | 0.055 | 0.839 | 0.870 | 84.999 | 9.230 | 0.000 |
| 2 | 0.422 | 0.002 | 17.852 | 96.183 | 5.460 | 23.713 | 0.000 |
| 3 | 0.182 | 0.044 | 0.761 | 4.593 | 298.819 | 300.705 | 0.000 |
| 4 | 6.823 | 7.074 | 31.279 | 18.736 | - | 320.212 | - |
| 5 | 2.350 | 2.048 | 0.921 | 0.937 | 9.825 | 11.013 | - |
| 6 | 0.109 | 0.013 | 0.894 | 0.938 | 0.975 | 0.850 | 0.000 |
| 7 | 0.146 | 0.384 | 14.939 | 0.881 | 0.973 | 338.277 | 0.000 |
| 8 | 0.626 | 1.745 | 6.170 | 9.299 | 0.500 | 0.683 | 0.000 |
| 9 | 1.612 | - | 0.909 | 0.697 | 0.728 | 0.961 | 0.000 |

Table B.21.: Goal Costs in Real-world Environment, Intersection 3 (Recording 30)

| Environment Seed | Motion Planning Approach | | | | | | |
|---|---|---|---|---|---|---|---|
| | Density Planner | | MPC | | MPC $r_{\text{tube}} = 0.5m$ | | Oracle |
| | ideal | bias | ideal | bias | ideal | bias | ideal |
| 0 | 0.271 | 0.932 | 0.855 | 57.533 | 13.202 | 0.949 | 0.000 |
| 1 | 0.458 | 0.062 | 0.953 | 5.035 | 49.776 | 58.762 | 1.452 |
| 2 | 0.413 | 0.036 | 0.835 | 22.305 | 4.852 | 10.631 | 0.000 |
| 3 | 2.434 | 1.708 | 3.267 | 5.318 | 0.207 | 0.246 | 0.000 |
| 4 | 0.400 | 1.004 | 0.881 | 5.049 | 153.172 | 271.948 | 0.000 |
| 5 | 0.003 | 0.010 | 35.114 | 0.722 | 943.510 | 5.376 | 0.000 |
| 6 | 1.188 | 0.481 | 0.814 | 35.650 | 279.015 | - | 0.000 |
| 7 | 0.286 | 0.825 | 0.549 | - | 0.552 | 0.771 | - |
| 8 | 0.084 | 0.204 | 20.247 | 13.297 | 342.498 | 644.610 | 0.000 |
| 9 | 4.102 | 5.688 | 0.844 | 0.790 | 543.894 | - | - |

Table B.22.: Input Costs in Real-world Environment, Intersection 1 (Recording 8)

| Environment Seed | Motion Planning Approach | | | | | | |
|---|---|---|---|---|---|---|---|
| | Density Planner | | MPC | | MPC $r_{\text{tube}} = 0.5m$ | | Oracle |
| | ideal | bias | ideal | bias | ideal | bias | ideal |
| 0 | 0.428 | 0.428 | 5.004 | 4.598 | - | - | 0.170 |
| 1 | 0.281 | 0.281 | 5.716 | - | 6.482 | - | 0.062 |
| 2 | 0.549 | 0.549 | 5.874 | 5.934 | 7.954 | - | - |
| 3 | 1.370 | 1.370 | 4.415 | - | - | - | 1.509 |
| 4 | 1.003 | 1.003 | 6.572 | 6.411 | - | - | - |
| 5 | 0.706 | 0.706 | 7.874 | - | - | 7.570 | 0.176 |
| 6 | 0.339 | 0.339 | 2.728 | - | - | 6.780 | - |
| 7 | 0.710 | 0.710 | 7.632 | 7.610 | 8.857 | 6.167 | 0.952 |
| 8 | 0.542 | 0.542 | 5.794 | 6.581 | 8.370 | 6.683 | 0.080 |
| 9 | 0.432 | 0.432 | 6.619 | 6.085 | 7.511 | 10.247 | 0.105 |

Table B.23.: Input Costs in Real-world Environment, Intersection 2 (Recording 26)

| Environment Seed | Motion Planning Approach | | | | | | |
|---|---|---|---|---|---|---|---|
| | Density Planner | | MPC | | MPC $r_{\text{tube}} = 0.5m$ | | Oracle |
| | ideal | bias | ideal | bias | ideal | bias | ideal |
| 0 | 0.440 | 0.440 | 1.275 | 3.453 | 5.072 | 7.439 | 4.747 |
| 1 | 0.927 | 0.927 | 4.404 | 5.185 | 11.410 | 10.608 | 6.518 |
| 2 | 0.840 | 0.840 | 7.194 | 5.196 | 7.856 | 7.335 | 0.458 |
| 3 | 0.675 | 0.675 | 7.178 | 6.021 | 6.690 | 6.228 | 1.320 |
| 4 | 1.503 | 1.503 | 4.370 | 3.912 | - | 7.354 | - |
| 5 | 1.025 | 1.025 | 2.331 | 2.236 | 5.240 | 6.124 | - |
| 6 | 0.523 | 0.523 | 5.175 | 4.560 | 5.729 | 5.231 | 2.517 |
| 7 | 1.016 | 1.016 | 5.463 | 7.613 | 6.673 | 8.929 | 2.519 |
| 8 | 0.740 | 0.740 | 3.675 | 3.502 | 0.813 | 5.355 | 2.011 |
| 9 | 0.552 | - | 6.548 | 6.516 | 2.644 | 6.033 | 3.707 |

Table B.24.: Input Costs in Real-world Environment, Intersection 3 (Recording 30)

| Environment Seed | Motion Planning Approach | | | | | | |
|---|---|---|---|---|---|---|---|
| | Density Planner | | MPC | | MPC $r_{\text{tube}} = 0.5m$ | | Oracle |
| | ideal | bias | ideal | bias | ideal | bias | ideal |
| 0 | 0.364 | 0.364 | 5.081 | 6.245 | 5.839 | 6.873 | 0.509 |
| 1 | 0.742 | 0.742 | 6.041 | 5.173 | 7.764 | 6.907 | 5.643 |
| 2 | 0.475 | 0.475 | 5.499 | 3.867 | 6.370 | 5.208 | 0.990 |
| 3 | 1.393 | 1.393 | 6.083 | 6.169 | 3.080 | 2.317 | 3.740 |
| 4 | 0.617 | 0.617 | 5.104 | 6.187 | 5.170 | 4.803 | 0.640 |
| 5 | 1.050 | 1.050 | 6.232 | 5.474 | 13.449 | 6.697 | 3.400 |
| 6 | 0.541 | 0.541 | 5.614 | 7.435 | 7.704 | - | 1.533 |
| 7 | 0.437 | 0.437 | 3.192 | - | 5.217 | 4.600 | - |
| 8 | 1.133 | 1.133 | 5.417 | 4.904 | 9.618 | 8.836 | 0.713 |
| 9 | 0.794 | 0.794 | 6.762 | 6.864 | 11.845 | - | - |

# List of Figures

# List of Tables

# Bibliography


[1] S. Bundesamt. "Verkehrsunfälle." In: *Fachserie 8 Reihe 7* (2019).

[2] M. Althoff. "Reachability Analysis and Its Application to the Safety Assessment of Autonomous Cars." PhD thesis. 2010.

[3] J. Lofberg. "Approximations of closed-loop minimax MPC." In: *42nd IEEE International Conference on Decision and Control*. Vol. 2. 2003, 1438–1442 Vol.2.

[4] S. Rakovic, E. Kerrigan, K. Kouramas, and D. Mayne. "Invariant approximations of the minimal robust positively Invariant set." In: *IEEE Transactions on Automatic Control* 50.3 (2005), pp. 406–410.

[5] F. Gruber and M. Althoff. "Computing Safe Sets of Linear Sampled-Data Systems." In: *IEEE Control Systems Letters* 5.2 (2021), pp. 385–390.

[6] A. Mesbah. "Stochastic Model Predictive Control: An Overview and Perspectives for Future Research." In: *IEEE Control Systems Magazine* 36.6 (2016), pp. 30–44.

[7] A. Wang, A. Jasour, and B. C. Williams. "Non-Gaussian Chance-Constrained Trajectory Planning for Autonomous Vehicles Under Agent Uncertainty." In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 6041–6048.

[8] C. Pek and M. Althoff. "Fail-Safe Motion Planning for Online Verification of Autonomous Vehicles Using Convex Optimization." In: *IEEE Transactions on Robotics* 37.3 (2021), pp. 798–814.

[9] M. Gulzar, Y. Muhammad, and N. Muhammad. "A Survey on Motion Prediction of Pedestrians and Vehicles for Autonomous Driving." In: *IEEE Access* 9 (2021), pp. 137957–137969.

[10] S. Hoermann, M. Bach, and K. Dietmayer. "Dynamic Occupancy Grid Prediction for Urban Autonomous Driving: A Deep Learning Approach with Fully Automatic Labeling." In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 2056–2063.

[11] J. Wu, J. Ruenz, and M. Althoff. "Probabilistic Map-based Pedestrian Motion Prediction Taking Traffic Participants into Consideration." In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. 2018, pp. 1285–1292.

[12] M. Koschi, C. Pek, M. Beikirch, and M. Althoff. "Set-Based Prediction of Pedestrians in Urban Environments Considering Formalized Traffic Rules." In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 2704–2711.

[13] S. Hacohen, S. Shoval, and N. Shvalb. "Probability Navigation Function for Stochastic Static Environments." In: *International Journal of Control, Automation and Systems* 17 (2019), 2097–2113.

[14] G. Aoude, B. Luders, and J. Joseph. "Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns." In: *Autonomous Robots* 35 (2013), pp. 51–76.

[15] Z. Huang, W. Schwarting, A. Pierson, H. Guo, M. Ang, and D. Rus. "Safe Path Planning with Multi-Model Risk Level Sets." In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 6268–6275.

[16] Y. Chen, T. T. Georgiou, and M. Pavon. "Optimal Steering of a Linear Stochastic System to a Final Probability Distribution, Part I." In: *IEEE Transactions on Automatic Control* 61.5 (2016), pp. 1158–1169.

[17] K. Okamoto and P. Tsiotras. "Optimal Stochastic Vehicle Path Planning Using Covariance Steering." In: *IEEE Robotics and Automation Letters* 4.3 (2019), pp. 2276–2281.

[18] J. Ridderhof, K. Okamoto, and P. Tsiotras. "Nonlinear Uncertainty Control with Iterative Covariance Steering." In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. 2019, pp. 3484–3490.

[19] Y. Chen. *Covariance Steering for Nonlinear Control-affine Systems*. Aug. 2021.

[20] Z. Yi, Z. Cao, E. Theodorou, and Y. Chen. "Nonlinear Covariance Control via Differential Dynamic Programming." In: *2020 American Control Conference (ACC)*. 2020, pp. 3571–3576.

[21] V. Krishnan and S. Martínez. "Distributed Optimal Transport for the Deployment of Swarms." In: *2018 IEEE Conference on Decision and Control (CDC)*. 2018, pp. 4583–4588.

[22] K. F. Caluya and A. Halder. "Finite Horizon Density Steering for Multi-input State Feedback Linearizable Systems." In: *2020 American Control Conference (ACC)*. 2020, pp. 3577–3582.

[23] K. F. Caluya and A. Halder. "Reflected Schrödinger Bridge: Density Control with Path Constraints." In: *2021 American Control Conference (ACC)* (2021), pp. 1137–1142.

[24] S. Haddad, K. F. Caluya, A. Halder, and B. Singh. "Prediction and Optimal Feedback Steering of Probability Density Functions for Safe Automated Driving." In: *IEEE Control Systems Letters* 5.6 (2021), pp. 2168–2173.

[25] Y. Chen and A. D. Ames. *Duality between density function and value function with applications in constrained optimal control and Markov Decision Process*. 2019.

[26] Z. Qin, Y. Chen, and C. Fan. "Density Constrained Reinforcement Learning." In: *Proceedings of the 38th International Conference on Machine Learning*. Vol. 139. PMLR, 2021, pp. 8682–8692.

[27] Y. Meng, Z. Qiu, M. T. B. Waez, and C. Fan. "Density of Reachable States and How to Use it for Safe Autonomous Motion Planning." In: *NASA Formal Methods Symposium (NFM)* (2022).

[28] W. Lohmiller and J.-J. E. Slotine. "On Contraction Analysis for Non-linear Systems." In: *Automatica* 34.6 (1998), pp. 683–696. ISSN: 0005-1098.

[29] Q.-C. Pham, N. Tabareau, and J.-J. Slotine. "A Contraction Theory Approach to Stochastic Incremental Stability." In: *IEEE Transactions on Automatic Control* 54.4 (2009), pp. 816–820.

[30] S. Singh, A. Majumdar, J.-J. Slotine, and M. Pavone. "Robust online motion planning via contraction theory and convex optimization." In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 5883–5890.

[31] H. Tsukamoto and S.-J. Chung. "Convex Optimization-based Controller Design for Stochastic Nonlinear Systems using Contraction Analysis." In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. 2019, pp. 8196–8203.

[32] M. Dresscher and B. Jayawardhana. "Prescribing transient and asymptotic behaviour of non-linear systems with stochastic initial conditions." In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. 2017, pp. 1957–1962.

[33] M. Dresscher and B. Jayawardhana. "Prescribing transient and asymptotic behaviour to deterministic systems with stochastic initial conditions." In: *International Journal of Control* 94.12 (2021), pp. 3506–3519.

[34] G. Terejanu, P. Singla, T. Singh, and P. D. Scott. "Uncertainty Propagation for Nonlinear Dynamic Systems Using Gaussian Mixture Models." In: *Journal of Guidance, Control and Dynamics* 31.6 (2008), pp. 1623–1633.

[35] Y. Xu, H. Zhang, Y. Li, K. Zhou, Q. Liu, and J. Kurths. "Solving Fokker-Planck equation using deep learning." In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 30.1 (2020).

[36] J. Zhai, M. Dobson, and Y. Li. "A deep learning method for solving Fokker-Planck equations." In: *Proceedings of the 2nd Mathematical and Scientific Machine Learning Conference*. Ed. by J. Bruna, J. Hesthaven, and L. Zdeborova. Vol. 145. Proceedings of Machine Learning Research. PMLR, 2022, pp. 568–597.

[37] W. Sun, J. Feng, J. Su, and Y. Liang. "Data driven adaptive Gaussian mixture model for solving Fokker-Planck equation." In: *Chaos* 32.3, 033131 (2022).

[38] V. Holubec, K. Kroy, and S. Steffenoni. "Physically consistent numerical solver for time-dependent Fokker-Planck equations." In: *Phys. Rev. E* 99 (3 2019), p. 032117.

[39] M. Ehrendorfer. "The Liouville Equation and Prediction of Forecast Skill." In: *Predictability and Nonlinear Modelling in Natural Sciences and Economics*. Ed. by J. Grasman and G. van Straten. Dordrecht: Springer Netherlands, 1994, pp. 29–44. ISBN: 978-94-011-0962-8.

[40] P. Sun, C. Colombo, M. Trisolini, and S. Li. "Comparison of continuity equation and Gaussian mixture model for long-term density propagation using semi-analytical methods." In: *Celestial Mechanics and Dynamical Astronomy* 134.22 (2022), pp. 1–30.

[41] Y. Meng, D. Sun, Z. Qiu, M. T. B. Waez, and C. Fan. "Learning Density Distribution of Reachable States for Autonomous Systems." In: (2021).

[42] I. R. Manchester and J.-J. E. Slotine. "Control Contraction Metrics: Convex and Intrinsic Criteria for Nonlinear Feedback Design." In: *IEEE Transactions on Automatic Control* 62.6 (2017), pp. 3046–3053.

[43] D. Sun, S. Jha, and C. Fan. "Learning Certified Control using Contraction Metric." In: *Conference on Robot Learning (CoRL)*. Nov. 1, 2020.

[44] K. Leung and I. R. Manchester. "Nonlinear stabilization via Control Contraction Metrics: A pseudospectral approach for computing geodesics." In: *2017 American Control Conference (ACC)*. 2017, pp. 1284–1289.

[45] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization." In: *CoRR* abs/1412.6980 (2015).

[46] L. Dubins. "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents." In: *American Journal of Mathematics* 79.3 (1957), pp. 179–202.

[47] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. "CasADi – A software framework for nonlinear optimization and optimal control." In: *Mathematical Programming Computation* 11.1 (2019), pp. 1–36.

[48] A. Waechter. "Getting Started with IPOPT in 90 minutes: Short tutorial." English (US). In: *Combinatorial Scientific Computing*. 2009.

[49] S. Yu, H. Chen, and F. Allgöwer. "Tube MPC scheme based on robust control invariant set with application to Lipschitz nonlinear systems." In: *2011 50th IEEE Conference on Decision and Control and European Control Conference*. 2011, pp. 2650–2655.

[50] R. Ghaemi, J. Sun, and I. V. Kolmanovsky. "Robust Control of Constrained Linear Systems With Bounded Disturbances." In: *IEEE Transactions on Automatic Control* 57.10 (2012), pp. 2683–2688.

[51] J. Bock, R. Krajewski, T. Moers, S. Runde, L. Vater, and L. Eckstein. "The inD Dataset: A Drone Dataset of Naturalistic Road User Trajectories at German Intersections." In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. 2020, pp. 1929–1934.

[52] M. Raissi, P. Perdikaris, and G. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations." In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991.