Fast and Safe Trajectory Optimization for Autonomous Mobile Robots Using Reachability Analysis

by

Sean K. Vaskov

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Mechanical Engineering)
in the University of Michigan
2021

Doctoral Committee:

Associate Professor Ramanarayan Vasudevan, Chair
Associate Professor Matthew Johnson-Roberson
Associate Professor Gabor Orosz
Professor Jeffrey Stein

Sean K. Vaskov
skvaskov@umich.edu
ORCID iD: 0000-0002-1054-2749

DEDICATION

To my parents Vinita and Steve, thank you for your unwavering love, encouragement, and support.

ACKNOWLEDGEMENTS

I would like to acknowledge my advisor Ram Vasudevan for all his mentorship and training, particularly early on in my PhD. Transitioning from industry back to academia was daunting at first, but at ROAHM lab I found the problems I want to solve and the research space I want to work in; I can't thank you enough. I would also like to acknowledge my labmates who I have worked with over the years Shreyas Kousik, Shannon Danforth, Partrick Holmes, Dan Bruder, Hannah Larson, Matthew Porter, Fan Bu, Ming-yuan Yu, Cyrus Anderson, Pengcheng Zhao, Jinsun Liu, Yifei Shao, Bohao Zhang. Working with and hanging out with you all has made the last five years a great chapter of my life.

I would like to acknowledge Ford motor company for funding most of my research at Michigan; particularly Eric Tseng, Shankar Mohan, and Vladimir Ivanovic, I would also like to thank Stewart Worrall and James Ward at the Australian Center for Field Robotics for giving Shreyas and me the opportunity to visit and collaborate during our winter and your summer. I would also like to thank Karl Berntorp and Rien Quirynen at Mitsubishi Electric Research Labs for a great summer internship. I really enjoyed the research we all did together, have learned a lot from each of you, and hope that we will keep in touch and collaborate in the future.

Lastly I would like to thank my family. Mom, Dad, and Ryan for providing me with so much support over the years and your frequent visits to Ann Arbor. Alex and Nicole, cannot imagine how grad school would have been without you here as well. Lastly, I would like to acknowledge my dog Nola for being a source of comfort and joy, and a getting me out of the house when I desperately needed study breaks. I owe more to you than you will ever know.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE

LIST OF TABLES

TABLE

LIST OF APPENDICES

ABSTRACT

Autonomous mobile robots (AMRs) can transform a wide variety of industries including transportation, shipping and goods delivery, and defense. AMRs must match or exceed human performance in metrics for task completion and safety. Motion plans for AMRs are generated by solving an optimization program where collision avoidance and the trajectory obeying a dynamic model of the robot are enforced as constraints.

This dissertation focuses on three main challenges associated with trajectory planning. First, collision checks are typically performed at discrete time steps. Second, there can be a nontrivial gap between the planning model used and the actual system. Finally, there is inherent uncertainty in the motion of other agents or robots.

This dissertation first proposes a receding-horizon planning methodology called Reachability-based Trajectory Design (RTD) to address the first and second challenges, where uncertainty is dealt with robustly. Sums-of-Squares (SOS) programming is used to represent the forward reachable set for a dynamic system plus uncertainty, over an interval of time, as a polynomial level set. The trajectory optimization is a polynomial optimization program over a space of trajectory parameters. Hardware demonstrations are implemented on a Segway, rover, and electric vehicle.

In a simulation of 1,000 trials with static obstacles, RTD is compared to Rapidly-exploring Random Tree (RRT) and Nonlinear Model Predictive Control (NMPC) planners. RTD has success rates of 95.4 % and 96.3 % for the Segway and rover respectively, compared to 97.6 % and 78.2 % for RRT and 0 % for NMPC planners. RTD is the only successful planner with no collisions. In 10 simulations with a CarSim model, RTD navigates a test track on all trials. In 1,000 simulations with random dynamic obstacles RTD has success rates of 96.8 % and 100 % respectively for the electric vehicle and Segway, compared to 77.3 % and 92.4 % for a State Lattice planner. In 100 simulations performing left turns, RTD has a success rate of 99 % compared to 80 % for an MPC controller tracking the lane centerline.

The latter half of the dissertation treats uncertainty with the second and/or third challenges probabilistically. The Chance-constrained Parallel Bernstein Algorithm (CCPBA) allows one to solve the trajectory optimization program from RTD when obstacle states are given as probability functions. A comparison for an autonomous vehicle planning a lane change with one obstacle shows an MPC algorithm using Cantelli's inequality is unable to find a solution when the obstacle's predictions are generated with process noise three orders of magnitude less than CCPBA. In

environments with 1-6 obstacles, CCPBA finds solutions in 1e-3 to 1.2 s compared to 1 to 16 s for an NMPC algorithm using the Chernoff bound. A hardware demonstration is implemented on the Segway.

The final portion of the dissertation presents a chance-constrained NMPC method where uncertain components of the robot model are estimated online. The application is an autonomous vehicle with varying road surfaces. In the first study, the controller uses a linear tire force model. Over 200 trials of lane changes at 17 m/s, the chance-constrained controller has a cost 86 % less than a controller using fixed coefficients for snow, and only 29 % more than an oracle controller using the simulation model. The chance-constrained controller also has 0 lateral position constraint violations, while an adaptive-only controller has minor violations. The second study uses nonlinear tire models on a more aggressive maneuver and provides similar results.

CHAPTER 1

Introduction

Research in autonomous mobile robots (AMRs) is of great interest to a variety of industries. In transportation, autonomous passenger vehicles have promised to reduce collisions and increase access to mobility for those who cannot drive personal vehicles. New task specific robots, such as those designed for food delivery, have been developed and may create new ways for goods to be distributed. In defense applications, similar types of delivery robots or autonomous vehicles may be used to eliminate the risk of human casualties from transportation missions.

In all of these applications, AMRs are expected to match or best human performance in both task completion and safety metrics. Human safety metrics are challenging to meet; consider the autonomous driving application for example. The NTSB stated in 2016 that for passenger vehicle occupants there were respectively 0.94, 121, and 500 fatalities, injuries, and crashes per 100 million vehicle-miles [SNR+18, Table 2-21]. For context, in the US, the average driver drives approximately 13,500 miles per year, meaning we may only expect to get into 2-4 accidents without injury, and 1 with injury in our entire lifetime. Furthermore, the accident statistics are further reduced for more structured or professional applications with trained users such as trucking–large truck occupants have fatality, injury, and crash rates of only 0.3, 13.2, and 174 per 100 million vehicle-miles [SNR+18, Table 2-23]. Meeting such stringent safety criteria is difficult for AMRs as they have to deal with large amounts of uncertainty in their own and others' motion. This seems trivial for expert human operators, but developing motion planning algorithms that can operate without mistake at the frequencies mentioned above is an active area of research.

This dissertation will focus on developing tools for provably and probabilistically safe motion planning. The application of focus is autonomous mobile robots, particularly ground vehicles. §1.1 gives a quick overview of motion planning and the associated challenges the dissertation will focus on. §1.2 describes the organization of the dissertation, giving a summary of the technical contributions. Lastly the notation used throughout the dissertation is presented in §1.3.

1.1 Overview

This section gives an overview of motion planning and challenges facing the state-of-the art that this dissertation addresses. §1.1.1 describes where the motion planner fits into the planning hierarchy that mobile robots use. §1.1.2 gives an overview of receding horizon motion planning and the challenges.

1.1.1 The Planning Hierarchy

Motion planning algorithms for AMRs typically employ a hierarchical strategy [KQCD15]. At the highest level, waypoints or route guidance goals are generated, usually with graph-based search algorithms [DSSW09]. The high-level planner does not use a dynamic model of the robot, so it is unable to ensure safety; however it drives performance and task accomplishment by providing a goal for the next level in the hierarchy. The middle level is referred to as the *motion planner* and is the focus of this dissertation. The motion planner produces a collision free trajectory that best achieves the goal given by the high level, and considers some dynamic model of the robot. This step is formulated as an optimization program; hence, in this dissertation, the terms motion planning and *trajectory optimization* are used interchangeably. The lowest level of the hierarchy consists of control systems that deliver commands (for example steering and power-train for an autonomous vehicle) to execute the trajectory produced by the middle level.

To adapt to changing environments, motion planning is typically done in a receding horizon framework, where trajectories of a finite time horizon, are planned and collision checked, while the robot executes the early portion of the previously planned trajectory [HGK10]. The receding horizon structure allows the motion planner to respond to new sensory information, however in order to ensure the robot can operate safely, strict requirements on the trajectory length, sensing horizon, and motion planner time limit must be met.

1.1.2 Challenges

The motion planner has to ensure that the vehicle is always able to find a *safe* plan. For mobile robots, safety is usually specified as collision avoidance. In the receding horizon framework, each plan typically ends in, or contains a fail-safe maneuver, that can be executed if a collision-free plan cannot be found at the next iteration. In the context of ground robots this is usually an explicit braking maneuver, or a terminal constraint ensuring that the ego vehicle will have sufficient distance from other vehicles to come to a stop [KTF+09, SSSS17, PA18, VLK+19]. In this framework, at a minimum, static obstacles must be sensed at a distance equal to the stopping distance, plus the distance traveled during one planning iteration. Since replanning at a faster rate

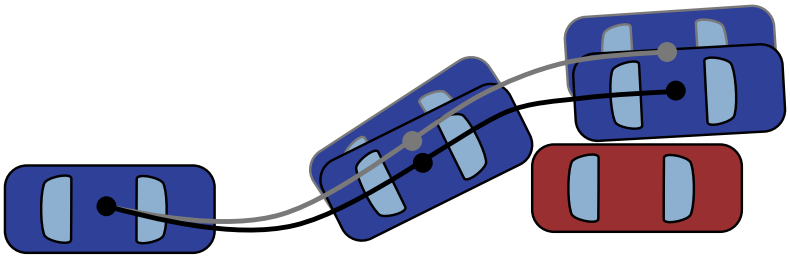


Figure 1.1: Illustration of two challenges facing motion planners for an autonomous vehicle. The ego vehicle (blue) is planning a lane change maneuver around a static obstacle (red). The predicted trajectory is shown in grey, the realized trajectory is shown in black. Positions for each are shown at 2 timesteps in the future. Performing a collision check that the gray time steps can result in a false negative due to: (1) motion of the vehicle between timesteps and (2) the gap between predicted and realized trajectory.

reduces the latter, one can see that keeping planning times low is critical to ensuring vehicles with finite sensor horizons can operate safely.

Motion planners must manage two tradeoffs between runtime and safety. First, the collision checker must certify that the trajectory is collision free over the planning time horizon; collision checking at discrete timesteps can miss collisions that occur in between them. Increasing the time discretization fineness increases the accuracy of the collision check, but also increases runtime. Second, predicting the ego robot's trajectory requires integration of a dynamic model; and using complex (more accurate) models increase runtime. To decrease runtime, simplified vehicle models can be used, however there will be a larger gap between the planned and realized trajectory; which can lead to collisions if unaccounted for. To avoid integrating a complex dynamic model online, some motion planners use precomputed libraries; however one must then balance richness of the library with runtime. Figure 1.1 provides an illustration of these two challenges with an autonomous passenger vehicle as an example. An additional challenge motion planners face is obstacles' positions and future behavior are uncertain. While this may be remedied in some applications with robot-to-robot connectivity, most widespread applications for autonomous robots cannot rely on communication with other obstacles or agents. For example, consider an autonomous vehicle lane keeping on a road, as as illustrated in Figure 1.2. The ego robot may be unsure if the adjacent, human-controlled vehicle will accelerate, decelerate or change lanes; although it is unlikely that the human will take actions that could cause a collision.

To summarize, this dissertation will contribute to three challenges in motion planning.

1. **Time discretization:** Performing collision checks at discrete timesteps introduces a tradeoff between runtime and safety. Collision checks should hold for all times in the planning time

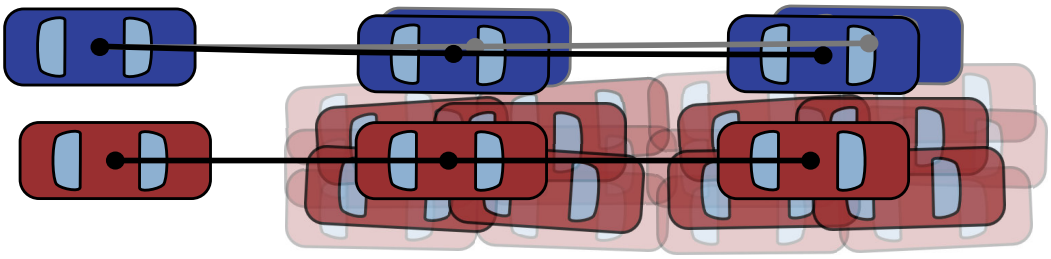


Figure 1.2: A third challenge is obstacle's (red) future motion may be uncertain. The ego vehicle (blue) is planning a lane keeping maneuver. The predicted trajectory is shown in grey, the realized trajectory is shown in black. Positions for the ego vehicle and obstacle are shown at 2 timesteps in the future. The transparent obstacles at future timesteps indicate a less likely outcome. A collision check must account for this uncertainty either robustly or probabilistically.

horizon.

2. **Integrating robot dynamic models:** Precomputing trajectories introduces a tradeoff between library richness and runtime. Simple dynamic models can be used for fast online planning, but they are inaccurate. Differences between the trajectory planning model and the actual robot motion must be accounted for.

3. **Uncertainty of obstacles' motion:** It is impossible to have perfect predictions of other actors' future motion. This uncertainty must be accounted for in the motion planner.

Uncertainty in challenges 2 and 3 can be dealt with *robustly* where all possible outcomes are accounted for, or *probabilistically* where the probability of a collision or unsafe event is bounded. Robust planners have stronger safety guarantees, but as shown in Figure 1.2 and discussed in [SSSS17], robust planning may be infeasible in some applications. Note that with respect to challenge 3, the focus of this dissertation is not on how to generate robust or probabilistic predictions of obstacles, but on developing trajectory optimization algorithms that can safely plan when given such predictions.

1.2 Organization and Contributions

This dissertation is organized as follows. Chapter 2 describes related work in the literature focused on motion planning and trajectory optimization with a focus on algorithms that view the world deterministically or treat uncertainty with the ego robot robustly. Chapters 3, 4, and 5 describe contributions from [KVJRV17, VSK+19, VKL+19, VLK+19, KVB+20] that address Challenges

1 and 2 robustly. Chapter 3 presents a Sums-of-Squares based method for reachable set computation of dynamic systems with affine disturbances. Chapter 4 describes the algorithm for online trajectory optimization, known as Reachability-based Trajectory Design (RTD), using the reachable set from Chapter 3. Chapter 5 presents demonstrations of RTD and comparisons of RTD against the state-of-the-art for both static and dynamic obstacles. Chapter 6 presents the Chance-constrained Parallel Bernstein Algorithm (CCPBA), a branch-and-bound algorithm developed to solve polynomial chance-constrained optimization programs. CCPBA is used addresses challenge 3 probabilistically, by solving the optimization program from RTD when the obstacles are represented as probability functions. Chapter 7 presents research on adaptive, chance-constrained, Nonlinear Model Predictive Control [VQB21]. This chapter contributes to challenge 2, however unlike the RTD framework, where the reachable set is computed offline; inaccuracies with the ego robot model are estimated online and incorporated into a chance-constrained NMPC program. Chapter 8 concludes the dissertation and provides general thoughts on future research directions.

1.3 Notation

This section gives an overview of notation used throughout the paper. §1.3.1 describes general mathematical notation. §1.3.2 describes notation relevant to trajectory optimization programs, primarily used in Chapters 4-7. §1.3.3 describes notation used in the dynamic models for the mobile robots in the dissertation.

1.3.1 Mathematical

Table 1.1 refers to mathematical notation used throughout the paper. Further details are provided below.

1.3.1.1 Points, Vectors, and Sets

Let $x := (x_1, x_2, \ldots, x_n) \in X \subseteq \mathbb{R}^n$ be a real variable of dimension n. A box $\mathbf{x}$ is an n-dimensional hypercube defined by $\{x \in \mathbb{R}^n \mid \underline{x}_i \leq x_i \leq \overline{x}_i,\ i \in \{1, \ldots, n\}\}$. A box is written as $\mathbf{x} := [\underline{x}_1, \overline{x}_1] \times \cdots \times [\underline{x}_n, \overline{x}_n] \subset \mathbb{R}^n$. The term subbox is used to signify that a box is a subset of a larger box. Boldface is used to distinguish subboxes from points or vectors. The width of a box in the i^{th} dimension is $\overline{x}_i - \underline{x}_i$. The maximum width of a box is given by $|\mathbf{x}| = \{\max(\overline{x}_i - \underline{x}_i) \mid i \in \{1, \ldots, n\}\}$. Numerical subscripts will usually refer to the index in a vector or list, or sometimes the timestep (in the case of a state space models). Numerical superscripts generally refer to the degree of a function or power of a variable. In certain instances subscripts and superscripts may be used to identify a particular item in a collection, as oppose to the dimension or degree. In cases

Category	Symbol	Description
Spaces	$\mathbb{N}$	natural numbers
	$\mathbb{R}^n$	n-dimensional real vector space
	$\mathsf{SE}(n)$	special-euclidean group
	$\mathbb{R}_{>0}$	real numbers greater than 0
	AC	absolutely continuous functions
	$C^{(1)}$	continuous functions (once differentiable)
	L_d	absolutely integrable functions
	$\mathbb{R}[x]$	ring of polynomials in x
Sets	$\mathcal{P}(X)$	power set of X
	∂X	boundary of X
	$\mathbf{x}$	box in X
	X^C	compliment of X
Functions and operators	δ_0	dirac delta
	$\|x\|_Q^2$	quadratic norm $x^\top Q x$
	$\mathbb{1}$	indicator function on $\mathbb{R}_{\geq 0}$
	dim(X)	dimension of X
	proj	projection operator
	$\circ$	Hadamard (element-wise) product
Logic	$\vee$	or
	$\wedge$	and
Probability	$\Pr$	probability function
	$\mathcal{F}$	event space
	Ξ	sample space
	ξ	outcome
	$\mathcal{N}$	normal distribution
	$\mathcal{GP}$	Gaussian process
	p	probability density function
	F	cumulative distribution function
	$\mathbb{E}$	expectation
	Var	variance

Table 1.1: Mathematical notation

where the context may be unclear parenthesis are used to indicate the former. For example $x^{(j)}$ would refer to the j^{th} item in a group.

1.3.1.2 Polynomials

A multi-index, J, is defined as $J := (j_1, j_2, \ldots, j_n) \in \mathbb{N}^n$ and the corresponding multi-power is defined as $x^J := (x_1^{j_1}, x_2^{j_2}, \ldots, x_n^{j_n})$. An inequality $J \leq D$ between 2 multi-indices, J and D, should be understood component wise. An n-variate polynomial in its monomial form can be written as

$$h(x) = \sum_{J \leq D} a_J x^J, \; x \in \mathbb{R}^n \tag{1.1}$$

with coefficients a_J and some multi-index $J \in \mathbb{N}^n$. The space of polynomials of degree $d \in \mathbb{N}$, with variable $x \in \mathbb{R}^n$, is $\mathbb{R}_d[x]$.

Definition 1. *$J \in \mathbb{N}^n$ is called the* multi-degree *of a polynomial h when each element of J is the maximum degree of the variable x_i out of all of the monomials of h. $d \in \mathbb{N}$ is called the* degree *of h when d is the maximum sum, over all monomials of h, of the powers of the variable x. That is, $d = ||J||_1$, where $|| \cdot ||_1$ is the sum of the elements of a multi-index.*

1.3.1.3 Probability

A probability space is denoted by a triple, $(\Xi, \mathcal{F}, \Pr)$. The sample space, Ξ, is the set of all possible outcomes. The collection of all events, $\mathcal{F}$, is a σ-algebra and is a set of subsets of Ξ. Finally, $\Pr : \mathcal{F} \to [0, 1]$ is a probability measure. Let $\mathbb{1} : \mathbb{R} \to \mathbb{R}$ denote the function that is equal to 1 whenever its argument is non-negative. The function F denotes the joint cumulative distribution function. For notational convenience if a box is the argument, F returns the cumulative distribution over the box, i.e. $F(\boldsymbol{\xi}) = \Pr(\xi \in \boldsymbol{\xi})$. When referring to quantities that may arise from and estimator or probabilistic algorithm, the accent $\hat{x}$ signifies that x is the related to average or mean of a point, vector or function, depending on context.

1.3.2 Trajectory Optimization

Table 1.2 notation pertains to the spaces, models, and functions relevant for reachable set computation and trajectory optimization. Examples of the notation used for RTD in Chapters 3-6 are given. Chapter 7, uses similar notation, although the spaces/domains of some of the functions are slightly different.

Category	Symbol	Description
Spaces	$t \in \mathbb{R}_{\geq 0}$	time
	T	time horizon
	$z_{\text{hi}} \in Z_{\text{hi}}$	state of high-fidelity (or simulation) model
	$z \in Z$	state of planning model
	$x \in X$	position states
	$k \in K$	trajectory parameter
	U	control input space
	u	control input or signal $u : T \to U$
	u_k	feedback controller for parameter k
Dynamics	f_{hi}	high-fidelity (simulation) model $f_{\text{hi}} : [0, T] \times Z_{\text{hi}} \times U \to \mathbb{R}^{n_{\text{hi}}}$
	$z_{\text{hi}}(t; z_{\text{hi},0}, k)$	trajectory of f_{hi} with initial condition $z_{\text{hi},0}$ and controller u_k
	f	planning model $f : [0, T] \times Z \times K \to \mathbb{R}^{n_Z}$
	g	affinely-appearing disturbance $g : [0, T] \times Z \times K \to \mathbb{R}^{n_Z}$
Reachable Sets	$\mathcal{X}_{\text{FRS}}$	Forward reachable set of positions over time horizon
	$\mathcal{Z}_{\text{TFRS}}$	Time-varying forward reachable set of states
Trajectory Optimization	$O_t^i \subset X$	obstacle i at time t
	P	prediction map $P : \mathbb{R}_{\geq 0} \to \mathcal{P}(X)$
	π_K	parameter projection map $\pi_K : \mathcal{P}(K) \to \mathcal{P}(X)$
	J	cost function $J : K \to \mathbb{R}$
	τ_{plan}	planning time

Table 1.2: Trajectory optimization notation

Symbol	Units	Description
(x, y)	m	position of center of mass
θ	rad	yaw
v	m/s	velocity
(v_x, v_y)	m/s	longitudinal and lateral (body-fixed) velocity
(a_x, a_y)	m/s	longitudinal and lateral (body-fixed) acceleration
δ	rad	wheel angle
(ω_f, ω_r)	rad/s	front and rear wheel speeds
$(F_{x,f}, F_{x,r})$	N	front and rear longitudinal tire force (wheel frame)
$(F_{y,f}, F_{y,r})$	N	front and rear lateral tire force (wheel frame)
(α_f, α_r)	rad	front and rear slip angles
(σ_f, σ_r)	rad	front and rear slip ratios
m	kg	mass
I	kg$\cdot$ m^2	yaw moment of inertia
l	m	wheelbase
(l_f, l_r)	m	distance from front and rear axle to center of mass
$(C_{y,f}, C_{y,r})$	N/rad	front and rear cornering stiffness
g	m/ s^2	gravitational acceleration
μ	1	friction coefficient

Table 1.3: Notation for ground robot models

1.3.3 Dynamic Models

The notation in Table 1.3 is commonly used for dynamic models of the robots in this dissertation. With the exception of the velocity $v = \sqrt{v_x^2 + v_y^2}$ and wheelbase $l = l_f + l_r$, dropping a subscript indicates a vector of the quantities; for example the vector of front and rear slip angles is written as $\alpha = [\alpha_f \ \alpha_r]^\top$. Note that some of the notation overlaps with that in Table 1.2. There are four instances the reader should be mindful of. First, x is used for 2D position states, but the traditional notation (x, y) is often used when plotting results or describing dynamic models. Second, the function g refers to an affinely appearing disturbance in Table 1.2, however when referring to normal forces in a vehicle model (primarily in §7.2) it is the gravitational constant. Third, F denotes the joint cumulative density function for a probability space in Table 1.1 and a function describing the tire force in Table 1.3. In Chapter 6, F always refers to the joint cumulative distribution function. In Chapter 7, F always refers to the tire force function. Fourth, a function v is defined in the reachability computations in Chapter 3 and as a constraint in the trajectory optimization in Chapter 6. "vel" is used to refer to the velocity in instances where this might be confusing.

CHAPTER 2

State-of-the-art

2.1 Motion Planning

Motion planning (also referred to as trajectory optimization) algorithms set up optimization programs to search for trajectories; where the cost function quantifies the trajectory's ability to accomplish the goal specified by high-level planner. Constraints in the program include checking that a planned trajectory is collision free, the trajectory obeys a vehicle dynamic model, and actuator limitations are satisfied. The trajectory optimization program is typically solved with one of three approaches: library-based, Rapidly-exploring Random Tree, or Model Predictive Control (MPC). This section focuses on deterministic planners, where the vehicle model is fixed. Tools for robust planning will be described in §2.2. Probabilistic and adaptive planners are discussed in §6.1 and §7.1.

2.1.1 Library-based

Library-based planners [MUDL11, GGL$^+$12, CPG17] precompute and store a library of trajectories, sometimes referred to as motion primitives. These planners are referred to as *discrete* planners, since they optimize over a countable set of trajectories. Online, sequences of primitives are collision checked against obstacles and evaluated for cost. An advantage to this approach is that the trajectories are computed offline, so they can be generated with a complex dynamic model, or be made to be smooth, resulting in little gap between the predicted and realized trajectory; however higher dimensions of initial conditions for the trajectories increases the size of the library, and the solve time of the planner. To control the size of the library, robots typically store and execute full maneuvers, i.e. they have limited ability to replan intermittently and they require the lower-level controller to accurately track the planned trajectories using PID, LQR, Linear Model Predictive Control or another control architecture. In some architectures the planning of a smooth path is conducted via graph search (blended with the high-level planner), then a speed profile is assigned

at the mid-level [KQCD15]. These limit the planning space of the robot greatly, as it cannot simultaneously optimize over directional and speed changes. Additionally, collision checks are typically performed at discrete time points along each trajectory [MUDL11]. Increasing the frequency of the collision check, or the amount of trajectories in the library also increases the runtime of the motion planner.

2.1.2 Rapidly-exploring Random Trees

Rapidly-exploring Random Trees (RRTs) randomly sample sequences of parameterized trajectories or control inputs to generate a tree that attempts to reach a goal. These are also stored in graph-like structure, however unlike the library-based approaches, the trajectories are not fixed apriori. The robot's trajectory is predicted using a dynamic or kinematic model, and the cost and obstacle avoidance constraints of the trajectory for each sample are evaluated [LKJ01, KTF+09]. The convergence guarantees of these algorithms are asymptotic, meaning the optimal solution is found as the number of samples tends to infinity. They suffer from a similar trade-off with respect to the collision checker and time discretization that the discrete planners face. Furthermore, increasing the complexity of the prediction model increases the runtime of the algorithm. As a result RRTs typically plan with simplified models, and require the lower-level controller to accurately track the planned trajectories similar to the library-based approaches. An advantage of the random planners is that they can easily run as anytime algorithms [Zil96]; once a feasible solution is found it can be returned as a suboptimal solution if a time limit is reached. However to this end, RRTs have been criticized for producing jagged solutions [KQCD15].

2.1.3 Model Predictive Control

Model Predictive Control algorithms for trajectory optimization [HGK10, PR14, BZAF19] use nonlinear, pseudo-spectral, or sequential quadratic programming to solve the trajectory optimization problem. In this formulation, the robot's dynamic model, input limits, and obstacle avoidance appear as constraints. Different from the discrete and random algorithms, derivatives of the cost and constraints are computed; which requires that the collision avoidance constraints are smooth. In single-level MPC architectures [FLJ+17], the motion planner uses a high dimensional robot model, so there is little onus placed on the lower-level controllers. For example in an autonomous vehicle, a single-level MPC architecture could optimize directly over inputs of acceleration and wheel angle; resulting in smoother trajectories than one would find with an RRT and reducing the complexity needed for the low-level controller. However similar to the discrete and random algorithms, the constraints are typically enforced at a set of discrete time points (referred to as collocation points). Additionally in standard implementations, model uncertainty is not taken into

account, and methods for robust model predictive control typically rely on linearizing the dynamics around a prespecified trajectory [GGC+14].

Compared to traditional linear MPC programs used in lower-level control, the nonlinear MPC (NMPC) programs for trajectory optimization take a nontrivial amount of time to solve. Several developments in the last few years have increased the tractability of NMPC for real-time applications. Real-time implementations using sequential quadratic programming [GZQ+20a], iteratively linearize the nonlinear program and solve a sequence of quadratic programs; however, they typically limit the number of iterations for real-time applications, which can introduce inaccuracies in terms of constraint satisfaction. Customizing nonlinear MPC implementations for tailored applications can also provide rapid solve times, for example [WSE20] uses a GPU-based implementation for a collision imminent steering problem with runtimes of less than 0.1 s. Recently developed, general NMPC solvers [FJSE20], have also produced solve times below 0.5 s for ground robot models, making their use in real-time applications tractable.

2.2 Safety Guarantees

The planning methods described in § 2.1 are unable to guarantee safety due to the challenges mentioned with time discretization and model inaccuracy. This section describes state-of-the-art methods for guaranteeing safety in motion planning and control. Safety guarantees can be incorporated at either the motion planner or low-level controller in the planning hierarchy; the advantages and disadvantages of each will be discussed.

2.2.1 Reachable Sets for Library-based Planners

To guarantee safety in the motion planner, a dynamic model with uncertainty is used to capture the difference between the planning model and actual robot. One must ensure the set of all possible future states of the robot are collision free; this set is referred to as the *forward reachable set* (FRS). The reachable set can be represented as a Zonotope, which can be propagated exactly through linear dynamic systems. For nonlinear systems, error incurred by the nonlinearities can be conservatively bounded. This approach has been applied as a way to verify planned trajectories of an autonomous vehicle [AD14]. An advantage of this approach is the zonotopes contain the reachable set over time intervals, allowing one to certify a trajectory is collision free over the entire time horizon. A disadvantage is that they can be conservative for nonlinear systems. In the context of trajectory planning this means that holding onto inputs or parameters in a way that is amenable to online optimization is difficult. Zonotopes have typically been limited to verification applications or discrete planners; although recent work [KHV19] has shown promise for online

optimization of spline trajectories.

Another approach called Funnel Libraries [MT17] uses a Sums-of-Squares (SOS) program to compute the reachable set of a vehicle dynamic model with uncertainty tracking a fixed reference trajectory. An advantage of this method over zonotopes, is that the SOS program is able to optimize both the reachable set and a polynomial feedback controller. Two limitations of this approach are that it has only been shown to work in static environments with a discrete motion planner, and the reachable sets are computed at discrete timesteps and passed to an off-the-shelf collision checker that does not provide safety guarantees.

The funnel library approach and zonotope approaches address challenge 2 in §1.1.2 robustly. The zonotope approach also addressed challenge 1 by constructing zonotopes containing the FRS for time intervals. Separate from the planning aspect, zontopes can be used to generate conservative predictions of obstacles [KA17], hence they can provide any of the motion planners in §2.1 with a way to address challenge 3 robustly.

2.2.2 Low-level Controller Design

To avoid using a discrete trajectory planner, one may chose to enforce safety at the low-level controller. Approaches involving the Hamilton Jacobi Bellman (HJB) equation [HCH$^+$17] have been developed to compute the worst case tracking error for a vehicle model with uncertainty tracking a low dimensional planning model. The advantage of this approaches are that any of the motion planners in § 2.1 can plan quickly with the low dimensional planning model; however in the HJB equation, the planning model is treated as a disturbance, so the tracking bounds are worst-case and conservative. A similar approach [FMM20] that uses control Lyapunov functions instead of the HJB equation has been developed; however, in this work, the Lyapunov functions are specifically designed for simplified robot models. The approaches address challenge 2 in §1.1.2 robustly, but they tend to be conservative or limited to low speed robots.

2.2.3 Barrier Functions

Control barrier functions (CBFs) [AXGT16] have been proposed for control affine systems to maintain safety over an infinite time horizon. They have been particularly popular in Adaptive Cruise Control and lane keeping for autonomous vehicles . They have also been applied to reinforcement learning (RL) tasks [COMB19] to maintain safety during learning, when the RL algorithm has limited understanding of the system. The CBF method uses a Lyapunov-style argument to render a desired constraint set control invariant; meaning an input can always be applied to keep the system within the safe set for all time. In [AXGT16], when the robot is running, a quadratic program uses the barrier function to modify the reference control input to keep the system in this

safe set. Barrier functions can be constructed analytically or with SOS [BMT12]. They can be developed to account for model or input uncertainty [Jan18, ATH+21]. Similar to the HJB approach, CBFs are flexible in a sense that the reference input can be given by any motion planner; and they additionally come with the benefit that the constraints are forward invariant for all time, so challenges 1 and 2 in §1.1.2 are both addressed. However, constructing the CBFs themselves often rely on apriori knowledge of the scenario and can be difficult to generalize to arbitrary and dynamic obstacles.

2.2.4 Contributions to Research Gap

Chapters 3-5 will present a planning and control framework based on reachability analysis. This subsection describes how the challenges in §1.1.2 are adressed and what my contributions in context of the state-of-the-art are. These chapters will focus on deterministic/robust planners. Probabilistic and adaptive planners are covered in Chapters 6 and 7.

In Chapter 3, I address challenge 2 by using Sums-of-Squares (SOS) programming to compute reachable sets, but will do so for parameterized trajectories. Holding onto the trajectory parameters means the planner will not be limited to a finite library like the approaches in §2.2.1. Performing the reachability computation with the trajectory parameters in mind, also renders the planner to be less conservative than those in §2.2.2. I address challenge 1 by computing FRSs over time intervals, and providing a provably safe method for representing static and dynamic obstacles in Chapter 4; which has yet to be done for the state-of-the-art motion planning application with SOS programming [MT17]. Finally the obstacle representation method will allow the planner to deal with arbitrary polygon obstacles online, overcoming the scenario specific limitations of the approaches in §2.2.3. Although I will not focus on how to specifically generate conservative predictions, the proposed planner can address challenge 3 by using conservative predictions from reachability-based perception modules such as [KA17].

CHAPTER 3

Parameterized Reachable Set Computation for Mobile Robots

3.1 Overview

This Chapter will describe how to compute reachable sets for mobile robots executing parameterized trajectories. For context, Figure 3.1 provides a simple illustration of what the reachable set contains and how it is useful for trajectory planning. The goal is to compute (offline) a reachable set that characterizes the reachable positions of the robot given a trajectory parameter. Then, when presented with obstacles online, an optimization program is set up where the constraints will eliminate unsafe parameters. This chapter covers the reachable set computation. The trajectory optimization portion is covered in Chapter 4. Examples of 2 hardware platforms in these sections, a Segway and Rover, are shown in Figure 3.2

This chapter is organized as follows. §3.2 describes dynamic models for the robot and reachable set computation. §3.3 defines the Forward Reachable Set (FRS) and optimization programs to compute it. §3.3.2 explains an Sums-of-Squares implementation to solve the aforementioned programs [KVJRV17, KVB$^+$20]. §3.4 explains a system decomposition technique to simplify the reachable set computation programs [KVB$^+$20]. §3.5 outlines programs to compute the FRS for time switching dynamics [VKL$^+$19, VLK$^+$19].

3.2 Preliminaries and Dynamic Models

In this chapter the reachable set for a mobile robot over a finite time horizon $T \in \mathbb{R}_{>0}$ will be computed. Let $f_{\text{hi}} : [0, T] \times Z_{\text{hi}} \times U \to \mathbb{R}^{n_{\text{hi}}}$ refer to the high-fidelity dynamic model that describes the robot

$$\dot{z}_{\text{hi}}(t) = f_{\text{hi}}(t, z_{\text{hi}}(t), u(\cdot)), \tag{3.1}$$

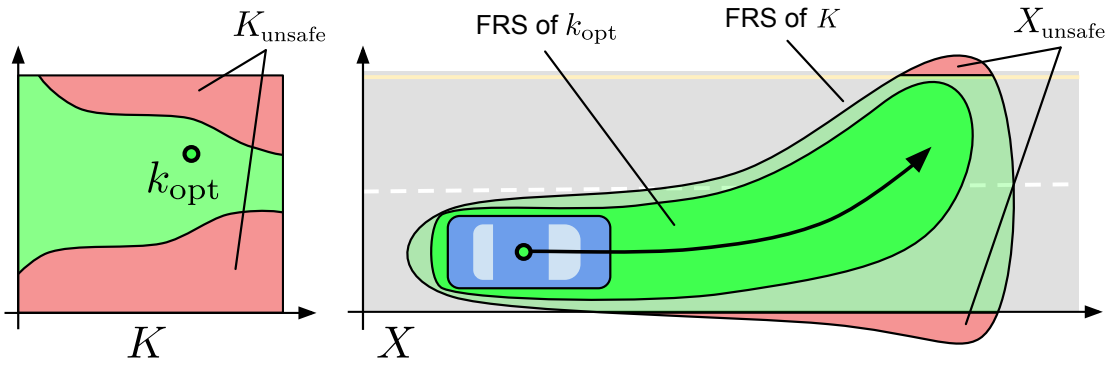
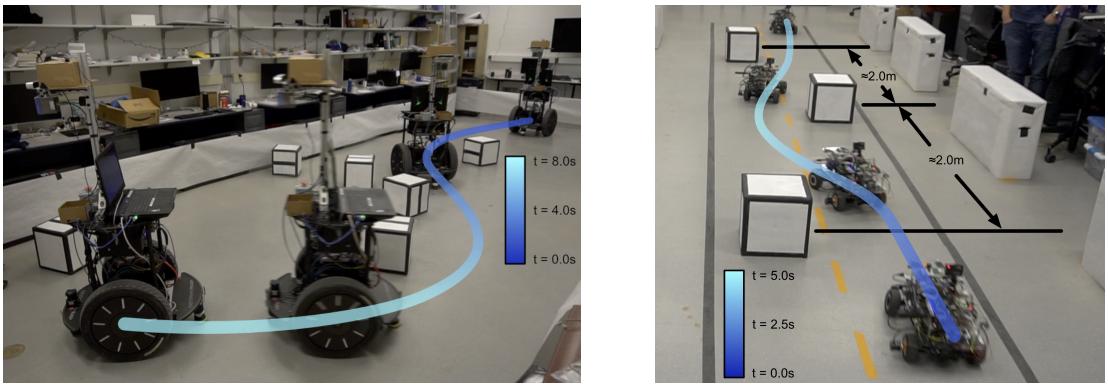


Figure 3.1: An illustration of the general approach to trajectory planning with reachable sets. The autonomous robot's trajectory parameter space K is on the left, and 2D position space, X, is on the right. The bell-shaped contour in X shows the total extent of the forward reachable set ($\mathcal{X}_{\text{FRS}}$), corresponding to the entirety of K. In X, areas are labeled as unsafe, corresponding to the labeled sets of trajectory parameters on the left. A safe parameter k_{opt} is selected in K, and the corresponding trajectory (the arrow) and corresponding subset of the FRS (the contour around the arrow) in X are shown on the right.



(a) (b)

Figure 3.2: Examples of two hardware platforms: the differential-drive Segway in Figure 3.2a and the car-like Rover in Figure 3.2b. Both robots safely traverse their respective scenarios despite error in each robot's ability to track planned trajectories. Videos of the robots are available at `https://youtu.be/FJns7YpdMXQ` for the Segway and `https://youtu.be/bgDEAi_Ewfw` for the Rover.

where u represents the input signal and $z_{\text{hi}} \in Z_{\text{hi}}$ is the vehicle state. The approach relies on planning over a space of parameterized trajectories using a second, lower dimensional, model that shares a common set of states $Z \subset Z_{\text{hi}}$. This model is called the trajectory-producing model and is described as

$$\begin{bmatrix} \dot{z}(t) \\ \dot{k}(t) \end{bmatrix} = \begin{bmatrix} f(t, z(t), k(t)) \\ 0 \end{bmatrix}. \tag{3.2}$$

For mobile robotic applications, the trajectory-producing model must at least contain the states that obstacles will appear in. For a trajectory parameterized by k, generated by (3.2), the robot selects inputs with a lower-level controller

$$u_k : [0, T] \times Z_{\text{hi}} \to U, \tag{3.3}$$

which yields the closed loop dynamics

$$\dot{z}_{\text{hi}}(t) = f_{\text{hi}}(t, z_{\text{hi}}(t), u_k(t, z_{\text{hi}}(t))). \tag{3.4}$$

The following assumptions will be used to ensure the computation of the reachable sets is tractable:

Assumption 2. *The dynamics f_{hi} from (3.1) are Lipschitz continuous in t, z_{hi}, and u. The dynamics f from (3.2) are Lipschitz continuous in t, z, and k.*

Assumption 3. *The sets U, Z_{hi}, Z, and K are compact. The robot's set of initial conditions are represented as a compact set $Z_{\text{hi},0} \subset Z_{\text{hi}}$ for the high-fidelity model, and $Z_0 = \text{proj}_Z(Z_{\text{hi},0})$ in the shared states of the trajectory-producing model.*

Examples of the high-fidelity model (3.1) and trajectory-producing model (3.2) are presented below.

Example 4. *Consider the Segway robot in Figure 3.2a, that can be described by a unicycle model as follows. Let $z_{\text{hi}} = [x, y, \theta, v, \dot{\theta}]^\top$ be the states, where x and y describe the robot's center of mass in 2D. Heading is θ, yaw rate is $\dot{\theta}$, and speed is v. The dynamics f_{hi} are*

$$\frac{d}{dt}\begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \\ v(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} v(t)\cos\theta(t) \\ v(t)\sin\theta(t) \\ \dot{\theta}(t) \\ \text{sat}_\alpha\Big(\beta_\alpha \cdot \big(u_1(t) - v(t)\big)\Big) \\ \text{sat}_\gamma\Big(\beta_\gamma \cdot \big(u_2(t) - \dot{\theta}(t)\big)\Big) \end{bmatrix}, \tag{3.5}$$

where the control input is $u = [u_1, u_2]^\top \in U \subset \mathbb{R}^2$*,* sat_α *(resp* sat_γ*) saturates the acceleration (resp. yaw rate change) input to keep it in an interval* $[\underline{\alpha}, \overline{\alpha}]$ *(resp.* $[\underline{\gamma}, \overline{\gamma}]$*), and* $\beta_\alpha,\ \beta_\gamma > 0$ *are constants found from system identification. The state space of the trajectory producing model is* $z = [x, y, \theta]^\top$*. The desired trajectories,* f*, are Dubin's paths*

$$\frac{d}{dt}\begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \end{bmatrix} = \begin{bmatrix} k_1 \cos\theta(t) \\ k_1 \sin\theta(t) \\ k_2 \end{bmatrix}, \tag{3.6}$$

where the parameters $k = [k_1, k_2]^\top \in K \subset \mathbb{R}^2$ *correspond to the desired speed and and yaw rate. The feedback controller used by the Segway is*

$$u_k(t, z_{\text{hi}}(t)) = \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} \tag{3.7}$$

[KVB+20] presents a trajectory producing model with further reduced dimensions:

Example 5. *Consider the Segway robot with high-fidelity model* (3.5) *and controller* (3.7)*. Assume the initial heading is* $\theta(0) = 0$*. The trajectory producing model* (3.2) *can be rewritten in 2 states given by*

$$\frac{d}{dt}\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} k_1 - k_2(y(t) - y_{c,0}) \\ k_2(x(t) - x_{c,0}) \end{bmatrix} \tag{3.8}$$

where $(x_{c,0}, y_{c,0})$ *are the initial position of the center of mass.*

3.2.1 Projection Operators

To better understand the relationship between various subspaces, projection operators, adapted from [CHT16, §III A, (15,16,18)] are defined. I will use the spaces Z and Z_{hi} as an example here, although the projections operators will also be used to relate subspaces within Z in § 3.4.

Definition 6. *The* projection operator $\text{proj}_Z : \mathcal{P}(Z_{\text{hi}}) \to \mathcal{P}(Z)$ *maps sets from a high dimensional space* Z_{hi} *to a lower dimensional subspace* $Z \subseteq Z_{\text{hi}}$*. For a set containing a single point,* $z_{\text{hi}} \in Z_{\text{hi}}$*,* proj_Z *is defined as:*

$$\text{proj}_Z(z_{\text{hi}}) = z, \tag{3.9}$$

where z *contains the components of* z_{hi} *that lie in subspace* Z_{hi}*. For a set,* $S_{\text{hi}} \subseteq Z_{\text{hi}}$*,* proj_Z *is*

defined as:

$$\text{proj}_Z(S_{\text{hi}}) = \Big\{ z \in Z \ : \ \exists \, z_{\text{hi}} \in S_{\text{hi}} \text{ s.t. } \text{proj}_Z(z_{\text{hi}}) = z \Big\}. \tag{3.10}$$

$\text{proj}^{-1} : \mathcal{P}(Z) \to \mathcal{P}(Z_{\text{hi}})$ *is defined as the* back-projection operator *from a subset, $S \subseteq Z$, to the full space, Z_{hi} is defined as:*

$$\text{proj}^{-1}(S) = \Big\{ z_{\text{hi}} \in Z_{\text{hi}} \ : \ \exists \, z \in S \text{ s.t. } \text{proj}_Z(z_{\text{hi}}) = z \Big\}. \tag{3.11}$$

System dynamics can be passed to the projection operators to select the dynamics in a subspace. For example, one may write $\text{proj}_Z(f_{\text{hi}}(\cdot))$ to mean the high-fidelity model's dynamics in the shared states Z, even though the range of f_{hi} does not return elements of $\mathcal{P}(Z_{\text{hi}})$; this is a minor abuse of notation because $\dim(Z_{\text{hi}}) = \dim(f_{\text{hi}}(\cdot))$. $x = \text{proj}_X(Z)$ recovers the states in z that correspond to 2D position.

3.2.2 Tracking Error

This section constructs bounds on the difference between the closed loop high-fidelity model (3.4) and the trajectory producing model (3.2). This is done because computing reachable sets with the high-fidelity model is typically intractable, but it is possible to compute a reachable set in the lower-dimensional state space of the trajectory producing model.

Assumption 7. *For each $i \in \{1, \ldots, n_Z\}$, there exists a bounded function $g_i : [0, T] \times Z \times K \to \mathbb{R}$ such that:*

$$\max_{z_{\text{hi}} \in A_z} |f_{\text{hi},i}(t, z_{\text{hi}}, k) - f_i(t, z, k)| \leq g_i(t, z, k), \tag{3.12}$$

for all $z \in Z$, $t \in [0, T]$, and $k \in K$ where $A_z := \{z_{\text{hi}} \in Z_{\text{hi}} \mid \text{proj}_Z(z_{\text{hi}}) = z\}$ is the set in which the high-fidelity model matches the trajectory-producing model in all the shared states. $g = [g_1, \ldots, g_{n_Z}]^\top$ is the tracking error function*. As with f and f_{hi} in Assumption 2, g is assumed to be Lipschitz continuous in t, z, and k.*

The existence of the tracking error function means that the error in the shared states subspace Z is bounded while the robot is tracking any desired trajectory given by (3.2). The tracking error function can be determined empirically by simulating the high-fidelity model or by applying Sums-Of-Squares (SOS) optimization techniques [Las09]. Figure 3.3 shows an example of g functions for the x and y states of the Segway described in Example 4.

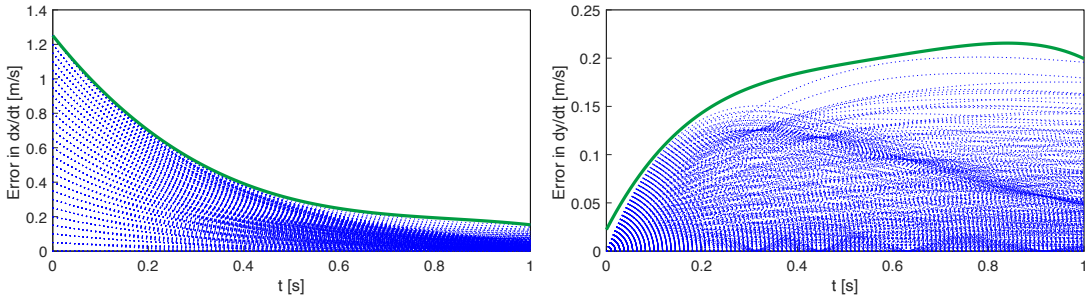


Figure 3.3: Error in $\dot{x}$ (left) and $\dot{y}$ (right) of the Segway's high-fidelity model (3.5) when tracking Dubins paths generated as in Example 4. The robot has a maximum yaw rate of 1 rad/s and a maximum speed of 1.25 m/s. Error is the expression $|f_{\text{hi},i}(t, z_{\text{hi}}, u_k(t, z_{\text{hi}})) - f_i(t, z, k)|$ in Assumption 7, where i selects the x and y components of z_{hi} and z. The dashed lines are example error trajectories created by sampling possible initial conditions. The solid lines represent the time-varying functions $g_x : [0, T] \to \mathbb{R}$ and $g_y : [0, T] \to \mathbb{R}$, which bound all of the error trajectories.

Remark 8. *Since the dynamics of the trajectory producing model and high-fidelity model are Lipschitz continuous by Assumption 2, and the time, state and parameter spaces are compact by Assumption 3, it is reasonable to assume that the tracking error can be bounded. In particular, if one is not confident that their robot can track trajectories closely, one can augment the tracking error function g with a large positive function. If computing a reachset for a hardware platform, or a system with uncertainty, where f_{hi} may not exactly describe the robot, a g function that satisfies Assumption 7 can still be found. This can be done by including uncertain parameters or disturbances in f_{hi}, checking to see if g satisfying* (3.12) *is conservative via sampling, and/or augmenting g with a large positive function. See §5.2.2 for an example of the second approach.*

Now I will describe how the error function is used to create a low-dimensional dynamic model that captures the behavior of the high-dimensional model. Let $L_d := L^1([0, T], [-1, 1]^{n_Z})$ denote the space of absolutely integrable functions from $[0, T]$ to $[-1, 1]^{n_Z}$. g is included in the trajectory-producing dynamics to create the *trajectory-tracking model* with dynamics:

$$\dot{z}(t, z(t), k, d) = f(t, z(t), k) + g(t, z(t), k) \circ d(t) \tag{3.13}$$

where $d \in L_d$, $\circ$ denotes the Hadamard product, and the state z of the trajectory-producing model is reused to emphasize that the trajectory-tracking model trajectories evolve in the state space Z. Note, d can be chosen to describe worst-case error behavior. Similarly, d can be used to make the trajectory-producing model "match" the high-fidelity model in the shared states:

Lemma 9. *Suppose $z_{\text{hi},0} \in Z_{\text{hi},0}$, $k \in K$, and $z_0 = \text{proj}_Z(z_{\text{hi},0})$. Then there exists $d \in L_d$ such that*

the high-fidelity model and the trajectory-tracking model agree on the shared state space Z, i.e.,

$$\text{proj}_Z\left(f_{\text{hi}}(t, z_{\text{hi}}(t), u_k(t, z_{\text{hi}}(t))\right) = \dot{z}(t, z(t), k, d(t)) \tag{3.14}$$

for all $t \in [0, T]$, where z_{hi} (resp. z) is a trajectory produced by (3.4) *(resp.* (3.13)*) with initial condition $z_{\text{hi},0}$ (resp. z_0).*

Proof. From Assumption 7 and (3.12), recall that g bounds the maximum possible absolute error in each shared state for all $t \in [0, T]$. Therefore, almost everywhere $t \in [0, T]$, $d(t) \in [-1, 1]^{n_Z}$ can be picked such that

$$\text{proj}_Z\left(f_{\text{hi}}(t, z_{\text{hi}}(t), u_k(t, z_{\text{hi}}(t))\right) - f(t, z(t), k) = g(t, z(t), k) \circ d(t). \tag{3.15}$$

Rearrange (3.15) to fulfill (3.14). □

Lemma 9 will come of use in § 3.3 as it enables one to perform reachability calculations with the low-dimensional model (3.13), that will certify the safety of the high-fidelity model (3.4).

In [VLK+19, KVB+20] I presented a less conservative version of the tracking error function where the error in the states as oppose to the dynamics is bounded.

Remark 10. *Consider a trajectory-producing model described by open-loop dynamics, for example as in Examples 4 and 5. Assume bounding functions $G : [0, T] \times K \to \mathbb{R}$ satisfying*

$$\max_{z_{\text{hi}} \in Z_{\text{hi},0}} |z_{\text{hi},i}(t; z_{\text{hi},0}, k) - z(t; z, k)| \leq G_i(\tau, k), \tag{3.16}$$

where $z_{\text{hi}}(t; z_{\text{hi},0}, k)$ indicates solutions of (3.4) *at time t with initial condition $z_{\text{hi},0}$ and parameter k, exist. Such a tracking error bound is less conservative, as it takes into account the integrated tracking error. For example, second order controllers may produce oscillations around a setpoint, the error bounds produced by* (3.12) *will lead to a more conservative FRSs as the dynamic error accumulates. To incorporate such a bound into the reachable set computations in §3.3, one can simply set $g(\cdot) = \frac{d}{dt}G(t, k)$, or alternatively see Appendix A.2.*

3.2.3 State Estimation Error

In a receding-horizon implementation, the robot plans a future trajectory while executing a previously planned trajectory. This requires bounding error associated with the initial condition set $Z_{\text{hi},0}$, which in the receding horizon implementation will be a predicted future state of the high-fidelity model. Error in this prediction is bounded with the following Assumption.

Assumption 11. *Let $k \in K$ be arbitrary and u_k the corresponding feedback controller as in* (3.3)*. Suppose the robot is at time t, with estimated state $z_{\text{hi},0} \in Z_{\text{hi},0}$, and note that τ_{plan} is the planning time. The robot's* future state prediction *z_{pred} at any $t' \in [t, t+\tau_{\text{plan}}]$ is given by forward-integrating the high-fidelity model to get the trajectory $z_{\text{hi},0} : [0, \tau_{\text{plan}}] \to Z_{\text{hi}}$:*

$$\begin{aligned} z_{\text{pred}}(t'; z_{\text{hi},0}, k) &= z_{\text{hi},0} + \\ &+ \int_0^{t'-t} f_{\text{hi}}(\tau, z_{\text{pred}}(\tau + t), u_k(\tau, z_{\text{pred}}(\tau + t)))d\tau. \end{aligned} \tag{3.17}$$

Assume the error between $z_{\text{pred}}(t'; z_{\text{hi},0}, k)$ and the actual state of the robot can be bounded by a constant ε_i for each $i \in \{1, \ldots, n_{\text{hi}}\}$. In other words, at the start of every planning iteration, there exist $\varepsilon \geq 0$ such that the state of the actual robot is within ε of its estimated state.

Note that Assumption 11 trivially satisfied by picking large ε. This error should be included in the initial condition of the FRS computation for hardware platforms. Additionally for states that do not appear in shared state space Z, the effects of the initial condition error should be included in the tracking error function defined in Assumption 7.

3.3 Forward Reachable Set Computation

This section defines the Forward Reachable Set (FRS) and discusses how we can use Sums-of-Squares programming to obtain a polynomial whose level set characterizes it. §3.3.1 introduces the concept of the forward reachable set at points in time, and defines an optimization program to find a function that characterizes the FRS. Then we define a reachable set that captures states across the continuous time horizon. §3.3.2 discusses a Sums-of-Squares (SOS) implementation of the optimization program developed in the prior sections. §3.3.3 discusses memory useage of the SOS implementation.

3.3.1 The Forward Reachable Set

The Forward Reachable Set (FRS) contains positions (in X) of the trajectory producing model that are reachable by a robot described by the high-fidelity model (3.4), over a time horizon T. This work focuses on ground applications where the robot's pose and environment can be represented in 2-D, i.e. the space $\mathbb{R}^2$ with coordinates denoted x and y.

Definition 12. *Let X denote the xy-*subspace of Z *with $\dim(X) = 2$. X refers to the* spatial coordinates *of the robot's body.*

In this and the next Chapter assume

Assumption 13. *The initial condition set* $\text{proj}_X(Z_0)$ *contains the footprint of the robot, and the* g *function, defined in Assumption 7 captures any error associated with rigid-body rotation.*

Remark 14. *Assumption 13 is the approach to incorporate the footprint that is applied in Chapters 3 and 4. Unfortunately this approach can be conservative for noncircular robots, especially when there is large uncertainty in the dynamics, or the time horizon* T *is long. A modified approach that decouples the footprint from the reachability computation is detailed in Appendix A.1.*

The reachable set will ultimately be used in collision checking (see Figure 3.1) which is a constraint that has to be enforced over continuous time intervals; meaning one wants to ensure the robot is collision free at any time $t \in [0, T]$. For collision checking against static obstacles it will prove useful to define an FRS for the entire time horizon as

$$\begin{aligned}\mathcal{X}_{\text{FRS}} = \Big\{ & (x,k) \in X \times K \mid \exists\, z_0 \in Z_0,\ t \in [0,T],\ \text{and } d \in L_d \\ & \text{s.t. } z = \tilde{z}(t),\ x = \text{proj}_X(z), \\ & \text{where } \dot{\tilde{z}}(\tau) = f(\tau, \tilde{z}(\tau), k) + g(\tau, \tilde{z}(\tau), k) \circ d(\tau) \\ & \text{a.e. } \tau \in [0,T] \text{ and } \tilde{z}(0) = z_0 \Big\}.\end{aligned} \tag{3.18}$$

The computation will rely upon a pair of linear operators, $\mathcal{L}_f, \mathcal{L}_g : AC\big([0,T] \times Z \times K\big) \to C\big([0,T] \times Z \times K\big)$ which act on a test function v as follows:

$$\mathcal{L}_f v(t,z,k) = \frac{\partial v}{\partial t}(t,z,k) + \sum_{i=1}^{n} \frac{\partial v}{\partial z_i}(t,z,k) f_i(t,z,k) \tag{3.19}$$

$$\mathcal{L}_g v(t,z,k) = \sum_{i=1}^{n} \frac{\partial v}{\partial z_i}(t,z,k) g_i(t,z). \tag{3.20}$$

With these operators, the FRS is computed by solving the following linear program, adapted from [MVTT14, Section 3.3, Program (D)]. The program has been altered for forward reacha-

bility and for uncertainty propagation in place of control synthesis.

$$
\begin{aligned}
\inf_{v,w,q} \quad & \int_{X\times K} w(x,k)\, d\lambda_{X\times K} && (D)\\
\text{s.t.} \quad & \mathcal{L}_f \tilde{v}(t,z,k) + q(t,z,k) \le 0, && (D1)\\
& \mathcal{L}_g \tilde{v}(t,z,k) + q(t,z,k) \ge 0, && (D2)\\
& -\mathcal{L}_g \tilde{v}(t,z,k) + q(t,z,k) \ge 0, && (D3)\\
& q(t,z,k) \ge 0, && (D4)\\
& -\tilde{v}(0,z,k) \ge 0, && (D5)\\
& w(x,k) \ge 0, && (D6)\\
& w(x,k) - v(t,z,k) - 1 \ge 0, && (D7)
\end{aligned}
$$

where, $x = \text{proj}_X(z)$ and $X = \text{proj}_X(Z)$. Constraints $(D1), (D2), (D3), (D4)$, and $(D7)$ apply for all $(t,z,k) \in [0,T] \times Z \times K$. Constraint $(D5)$ applies for all $(z,k) \in Z_0 \times K$. Constraint $(D6)$ is enforced on $X \times K$. The infimum is taken over $(v,w,q) \in C^1([0,T] \times Z \times K) \times C(X \times K) \times C([0,T] \times Z \times K)$. Theorem 3 of [MVTT14] show that feasible solutions to (D) conservatively approximate $\mathcal{X}_{\text{FRS}}$. This result is adapted in the following lemmas.

Lemma 15. *If (v,w,q) satisfies the constraints in (D), then v is non-positive and decreasing along trajectories of the trajectory-tracking system* (3.13)*.*

Proof. Notice that $v(0,z_0,k) \le 0$ for all $z_0 \in Z_0$ and $k \in K$ by $(D5)$. So, for any $t \in [0,T]$, $k \in K$, and $d \in L_d$:

$$
\begin{aligned}
v(t,z(t),k) = \; & v(0,z(0),k) + \int_0^t (\mathcal{L}_f v(\tau,z(t),k))\, d\tau + \int_0^t (\mathcal{L}_g v(t,z(\tau),k) \circ d(\tau))\, d\tau && (3.21)\\
\le \; & v(0,z(0),k) + \int_0^t (\mathcal{L}_f v(\tau,z(\tau),k))\, d\tau + \int_0^t q(\tau,z(\tau),k) d\tau && (3.22)\\
\le \; & v(0,z(0),k), && (3.23)
\end{aligned}
$$

where (3.21) follows from the Fundamental Theorem of Calculus; (3.22) follows from (D_2) and (D_3); and (3.23) follows from (D_1). □

Lemma 16. *[MVTT14, Theorem 4] Let (v,w,q) be a feasible solution to (D). The 1-superlevel set of w contains $\mathcal{X}_{\text{FRS}}$. Furthermore, there is a sequence of feasible solutions to (D) whose second component w converges from above to an indicator function on $\mathcal{X}_{\text{FRS}}$ in the L^1-norm and almost uniformly.*

Proof. The first statement is given from constraint (D_7); which provides that the 1-superlevel set of any feasible w contains $\mathcal{X}_{\text{FRS}}$. See [MVTT14] for a proof of the second. □

3.3.2 Sums-of-Squares Implementation

The program (D) can be implemented using Sums-of-Squares programming (SOS). To do so, the following assumptions are required:

Assumption 17. *The functions f and g are polynomials of finite degree in $\mathbb{R}[t, z, k]$.*

Note that, to fulfill Assumption 7, if f is Taylor-expanded to be a polynomial, then g must bound the error introduced by the Taylor-expansion.

Assumption 18. *The sets $K,\ Z$, and Z_0 have semi-algebraic representations:*

$$K = \{k \in \mathbb{R}^{n_K} \mid h_{K_i}(k) \geq 0,\ \forall\, i = 1, ..., n_K\} \tag{3.24}$$

$$Z = \{z \in \mathbb{R}^{n_Z} \mid h_{Z_i}(z) \geq 0,\ \forall\, i = 1, ..., n_Z\} \tag{3.25}$$

$$Z_0 = \{z \in Z \mid h_{0_i}(z) \geq 0,\ \forall\, i = 1, ..., n_0\} \tag{3.26}$$

where $h_{K_i} \in \mathbb{R}[k]$ and $h_{0_i}, h_{Z_i} \in \mathbb{R}[z]$. Since X and X_0 are projected (as in Definition 6) from semi-algebraic sets, they can also be represented semi-algebraically:

$$X = \{p \in \mathbb{R}^2 \mid h_x(p) \geq 0,\ h_y(p) \geq 0\} \tag{3.27}$$

$$X_0 = \{p \in X \mid h_{x_0}(p) \geq 0,\ h_{y_0}(p) \geq 0\}. \tag{3.28}$$

Assumption 18 is not prohibitive since common boxes and ellipses, and even non-convex sets, have semi-algebraic representations (see, e.g., [MVTT14]). Typically, Z, and Z_0 are box- or ellipse-shaped. The parameter space K can be represented by a box or ellipse; more complex restrictions of the parameters can be enforced in the online optimization program described in Chapter 4. Also note that, for the SOS program posed next, there must exist $N \in \mathbb{N}$ such that, for any $q = (t, z, z_0, k) \in [0, T] \times Z \times Z_0 \times K$, the value of $N - \|q\|_2^2 \geq 0$ [Las09, Theorem 2.15]. This is trivially satisfied since $[0, T]$, $Z,\ Z_0$, and K are compact by Assumption 3.

(D) can be solved with a sequence of convex SOS programs indexed by $l \in \mathbb{N}$ by relaxing the continuous function in (D) to polynomial functions with degree truncated to $2l$. The inequality constraints in (D) then transform into SOS constraints, so (D) becomes a Semi-Definite Program (SDP) [Par00]. To formulate this problem, let $h_T = t(T - t)$, and $H_T = \{h_T\}$. Recalling the definitions in Assumption 18, for Z, Z_0, and K, collect the polynomials that represent them in the sets $H_Z = \left\{h_{Z_1}, \ldots, h_{Z_{n_Z}}\right\}$, $H_{Z_0} = \left\{h_{0_1}, \ldots, h_{0_{n_0}}\right\}$, $H_K = \left\{h_{K_1}, \ldots, h_{K_{n_K}}\right\}$, and $H_X = \{h_x, h_y\}$.

Let $Q_{2l}(H_T, H_Z, H_K) \subset \mathbb{R}_{2l}[t, z, k]$ be the set of polynomials $p \in \mathbb{R}_{2l}[t, z, k]$ (i.e., of total

degree less than or equal to $2l$) expressible as:

$$p = s_0 + s_1 h_T + \sum_{i=1}^{n_Z} s_{i+2} h_{Z_i} + \sum_{i=1}^{n_K} s_{i+n_Z+2} h_{K_i}, \tag{3.29}$$

for some polynomials $\{s_i\}_{i=0}^{n_Z+n_K+1} \subset \mathbb{R}_{2l}[t,z,k]$ that are SOS of other polynomials. Note that every such polynomial is non-negative on $[0,T] \times Z \times K$ [Las09, Theorem 2.14]. Similarly, define $Q_{2l}(H_{Z_0}, H_K) \subset \mathbb{R}_{2l}[z,k]$, and $Q_{2l}(H_X, H_K) \subset \mathbb{R}_{2l}[x,k]$, where $x = \text{proj}_X(z)$.

Employing this notation, the l^{th}-order relaxed SOS programming representation of (D), denoted (D^l), is defined as follows:

$$\begin{aligned}
\inf_{v^l, w^l, q^l} \quad & y_{X\times K}^T \text{vec}(w^l) & & & (D^l) \\
\text{s.t.} \quad & -\mathcal{L}_f v^l - q^l & & \in Q_{2d_f}(H_T, H_Z, H_K) & (D^l 1) \\
& \mathcal{L}_g v^l + q^l & & \in Q_{2d_g}(H_T, H_Z, H_K) & (D^l 2) \\
& -\mathcal{L}_g v^l + q^l & & \in Q_{2d_g}(H_T, H_Z, H_K) & (D^l 3) \\
& q^l & & \in Q_{2l}(H_T, H_Z, H_K) & (D^l 4) \\
& -v^l(0,\ \cdot\ ) & & \in Q_{2l}(H_{Z_0}, H_K) & (D^l 5) \\
& w^l & & \in Q_{2l}(H_X, H_K) & (D^l 6) \\
& w^l + v^l - 1 & & \in Q_{2l}(H_T, H_Z, H_K), & (D^l 7)
\end{aligned}$$

where the infimum is taken over the vector of polynomials $(v^l, w^l, q^l) \in \mathbb{R}_{2l}[t,z,k] \times \mathbb{R}_{2l}[x,k] \times \mathbb{R}_{2l}[t,z,k]$, with $x = \text{proj}_X(z)$. The vector $y_{X\times K}$ contains moments associated with the Lebesgue measure $\lambda_{X\times K}$, so $\int_{X\times K} w(x,k) d\lambda_{X\times K} = y_{X\times K}^T \text{vec}(w)$ for $w \in \mathbb{R}_{2l[x,k]}$ [MVTT14]. The numbers d_f and d_g are the smallest integers such that $2d_f$ and $2d_g$ are respectively greater than the total degree of $\mathcal{L}_f v^l$ and $\mathcal{L}_g v^l$. To implement (D^l), consider the dual program, which is an SDP [Las09].

Remark 19. *It can be shown that Lemma 15 holds for functions that satisfy the constraints of (D^l) [MVTT14, Theorem 6]. Additionally, one can apply the last constraint in (D^l) to prove that the 1-superlevel set of any feasible w^l is an outer approximation to $\mathcal{X}_{\text{FRS}}$ [MVTT14, Theorem 7]. Furthermore, one can prove that w^l converges from above to an indicator function on $\mathcal{X}_{\text{FRS}}$ in the L^1-norm [MVTT14, Theorem 6].*

An example of the 1 superlevel set of w^5 for the Segway dynamics from Example 5 is shown in Figure 3.4

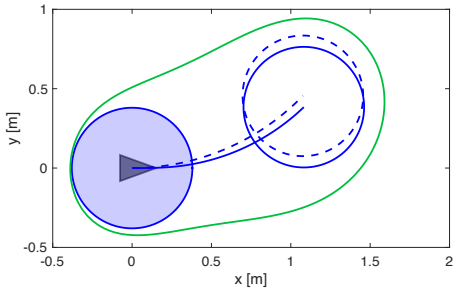


Figure 3.4: The Segway robot, as in Example 4, tracking trajectories planned in the xy-subspace X using the trajectory-producing model Example 5. The robot begins with $[x(0), y(0)]^\top = [0, 0]^\top$ and the initial heading $\theta(0) = 0$ rad pointing to the "right." The robot has a circular footprint with radius 0.38 m, and the initial state is $[x(0), y(0), \theta(0), v(0), \dot{\theta}(0)]^\top = [0 \text{ m}, 0 \text{ m}, 0 \text{ rad}, 1.5 \text{ m/s}, 0.0 \text{ rad/s}]^\top$, plotted in X as the solid circle on the left. The desired speed (k_1) is $v_{\text{des}} = 1.5$ m/s; the desired yaw rate (k_2) is $\dot{\theta}_{\text{des}} = 1.0$ rad/s. The desired trajectory, with time horizon $T = 0.8$ s, is shown in dashed blue, with the robot's footprint plotted at the end. The high-fidelity model trajectory, and corresponding footprint at time T is shown in solid blue.

3.3.3 Memory Useage

The section describes the memory required to implement (D). For higher-dimensional systems, the large memory requirement motivates the system decomposition approach in Section 3.4. In this paper, the FRS is computed with Spotless [TPM13], a MATLAB-based SOS toolbox. Spotless transforms the SOS optimization program into an Semidefinite Program (SDP), which is solved with MOSEK [Mos10]. As the degree l increases, the approximation of the FRS becomes a provably less conservative outer approximation of $\mathcal{X}_{\text{FRS}}$ [MVTT14, Theorem 7].

However, solving (D^l) is memory intensive, as the monomials of each polynomial are free variables; and a polynomial of degree $2l$ and dimension n has $\binom{2l+n}{n}$ monomials. The memory required by (D^l) grows as $\mathcal{O}((n+1)^l)$ for fixed l and $\mathcal{O}(l^{n+1})$ for fixed n [MVTT14, Section 4.2]. Furthermore each free variable is stored as a 64-bit double, and MOSEK computes the Hessian of each SOS constraint [Mos10, Section 11.4], which is proportional to the number of free variables squared (see, e.g., [NW06, Chapter 14]. To estimate the amount of free variables generated for Program (D^l), one can sum up the monomials in each decision variable polynomial. These consist of the polynomials v^l, w^l, q^l which are degree $2l$, and the s polynomials for each semialgebraic set, defined in (3.29), whose degree are specified in (D^l).

The Segway model in Example 5 has dimension 5. Solving (D^5) for this system requires approximately 1.4×10^5 free variables. The highest dimension system for which I have computed reachable sets for is the car-like rover, shown in Figure 3.2b, which is described in the following

section. It has a 7-dimensional state space model. The program (D^3) for this system requires approximately 1.1×10^5 free variables, and MOSEK used 504 GB of RAM (i.e., memory). A computer with 3.5 TB of RAM was unable to solve (D^4), which has approximately 3.8×10^5 free variables. This drastic increase in memory required as dimension increases motivates the system decomposition approach in Section 3.4, where separate, lower dimension FRS's are computed for subsystems, then combined into the full dimension system with another SOS program.

3.4 System Decomposition

Motivated by the memory issues presented in Section 3.3.3, this section presents a method to apply the FRS computation to higher-dimensional systems. This section is broken into four parts. First, the Rover robot is presented as an example system for which the method presented in Section 3.3.2 is intractable (§3.4.1). Second, I describe a system decomposition that makes computing an FRS tractable for each subsystem of the decomposed system (Section 3.4.2). Third, I show how to reconstruct an FRS of the full system from FRS's computed for the subsystems (Section 3.4.3). Lastly, the SOS implementation and memory useage are discussed in 3.4.4. The casual reader can examine Example 20, Definition 22, Example 23, and Theorem 24 to understand the primary results from this section.

As a further motivation, consider Example 4, where the trajectory-producing model creates arcs with constant speed and yaw rate. In some applications, it is beneficial to plan with more complicated trajectories. For example a passenger vehicle on a road would plan lane change, lane keeping, and lane return maneuvers, requiring a higher-dimensional model than the one that produces arcs. As noted in Section 3.3.3, for fixed relaxation degree l, increasing the dimension n of the trajectory-producing model increases memory usage of (D^l) as $\mathcal{O}((n+1)^l)$ [MVTT14, Section 4.2].

This section adapts a general method from [CHT16] and [CHV$^+$17] for computing backwards reachable sets by system decomposition. I adapt the method to forward reachability, illustrate how to apply SOS programming, and analyze the memory savings that result from using system decomposition. This type of decomposition applies when the robot's dynamic model can be split into subsystems of lower dimension. For example, the Segway and Rover model presented in this work, or the quadrotor models presented in [CHT16, KHV19]. Note that the recovery of the exact forward reachable set is not always possible with this approach. However, the resulting reachable set is guaranteed to be an overapproximation; hence is useful for the presented application of collision checking. This section focuses on the case with two subsystems, though the approach generalizes to any finite number. As per [CHT16] and [CHV$^+$17], after separating the system, a reachable set is computed for each subsystem, then the reachable set for the full system is constructed by

intersecting the subsystem reachable sets.

3.4.1 An Example System

This section describes a practical example of trajectory-producing dynamics with a system dimension that makes computing the FRS intractable.

Example 20. *Recall the Rover from Figure 3.2b. This robot uses the following bicycle model as the trajectory-producing model* (3.2)*, with states $z = [x, y, \theta]^\top \in Z \subset \mathbb{R}^3$ where x and y track the center of mass.*

$$\frac{d}{dt}\begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \end{bmatrix} = \begin{bmatrix} k_1 \cos(\theta(t)) - l_r \dot{\theta}(t,k) \sin(\theta(t)) \\ k_1 \sin(\theta(t)) + l_r \dot{\theta}(t,k) \cos(\theta(t)) \\ \dot{\theta}(t,k) \end{bmatrix} \tag{3.30}$$

$$\dot{\theta}(t,k) = -2\frac{T_h k_2 - k_3}{T_h^2} t + k_2 \tag{3.31}$$

where $\dot{\theta}$ is yaw rate, k_1 is longitudinal speed, and l_r is the distance from the robot's rear-wheel to center of mass. The trajectory parameters, $k \in K \subset \mathbb{R}^3$, produce lane change, lane keeping, and lane return maneuvers for an autonomous car driving on a straight road. T_h is the time required to complete a lane change; k_2 determines the initial yaw rate of the trajectory; and k_3 is the final heading of the trajectory.

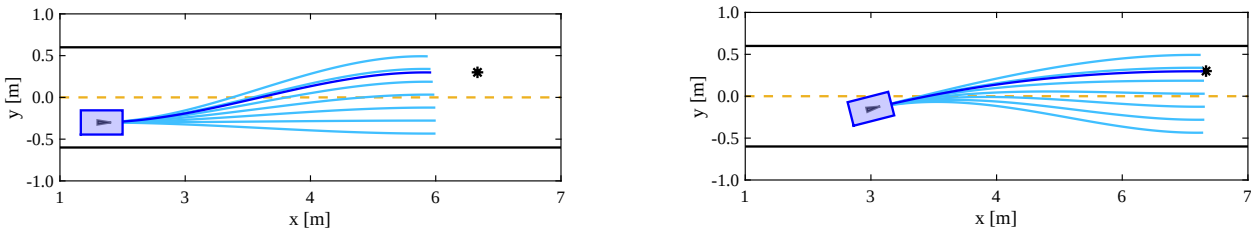
To understand this parameterization, integrate (3.31) *over time with initial condition $\theta(0) = 0$ to get the robot's heading:*

$$\theta(t) = -\frac{T_h\, k_2 - k_3}{T_h^2} t^2 + k_2 t \tag{3.32}$$

Notice that k_2 determines the final lateral displacement of the robot, and setting k_3 to the difference between the road and robot heading will create trajectories that align the robot with the road. Sample maneuvers generated by (3.30) *and* (3.31) *are depicted in Figure 3.5. This parameterization captures lane change, lane keeping, and lane return maneuvers. The total dimension of* (3.30)*, including time, is $n = n_Z + n_K + 1 = 7$, which is intractable for the FRS computation as in Section 3.3. However, the full system* (3.30) *is separable into "self-contained subsystems."*

3.4.2 Self-contained Subsystems

This section describes how to decompose the trajectory-producing model (3.2) into two subsystems. I follow the methodology introduced by [CHV+17, Section III A], adapting the notation and



(a) (b)

Figure 3.5: Sample of lane change trajectories generated by (3.30) with the control law in (3.31). The rectangle containing a triangle "pointer" represents the Rover and its initial heading. Initial headings of 0.0 and 0.25 are shown in subfigures (a) and (b), respectively. In subfigure (a), the Rover is driving straight in its lane and the sample trajectories consists of lane keeping and lane change maneuvers. In subfigure (b), the Rover has begun a lane change, and the sample trajectories consist of lane return maneuvers, and trajectories that complete a lane change. The parameters used are $T_h = 2$ s and $k_1 = 2$ m/s, with $k_3 = 0.0$ in subfigure (a) and $k_3 = -0.25$ in (b). The light trajectories are generated with a sample of values of k_2 and plotted over a time horizon of 2 s. The dark trajectory is the optimal trajectory to reach a desired waypoint, shown as an asterisk.

dynamics to system (3.2), which is refered to as the *full system*. Let the state, $z \in Z$, be partitioned as $z = (z_1, z_2, z_s)$ with $z_1 \in \mathbb{R}^{n_1}$, $z_2 \in \mathbb{R}^{n_2}$, $z_s \in \mathbb{R}^{n_s}$, $n_1, n_2 > 0$, $n_s \geq 0$, and $n_1 + n_2 + n_s = n_Z$. Note, the notation (z_1, z_2, z_s) (as opposed to $[z_1^\top, z_2^\top, z_s^\top]^\top$) is used in this section for readability. The states z_1 and z_2 belong to subsystems 1 and 2, respectively, and the states in z_s belong to both subsystems. Therefore, the dynamics can be written:

$$\begin{aligned} \dot{z}_1(t) &= f_1(t, z_1(t), z_2(t), z_s(t), k) \\ \dot{z}_2(t) &= f_2(t, z_1(t), z_2(t), z_s(t), k) \\ \dot{z}_s(t) &= f_s(t, z_1(t), z_2(t), z_s(t), k) \\ \dot{k}(t) &= 0. \end{aligned} \tag{3.33}$$

Next define the subsystem states and spaces $z_1 = (z_1, z_s) \in Z_1 \subset \mathbb{R}^{n_1+n_s}$ and $z_2 = (z_2, z_s) \in Z_2 \subset \mathbb{R}^{n_2+n_s}$. The subspaces Z_1 and Z_2 are compact and have semi-algebraic representations. Just as in Assumption 18, the initial conditions and state space of subsystem i are defined as:

$$Z_{0,i} = \left\{ z_i \in \mathbb{R}^{n_i+n_s} : \; h_{0_j}(\text{proj}^{-1}(z_i)) \geq 0 \; \forall \, j = 1, .., n_0 \right\} \tag{3.34}$$

$$Z_i = \left\{ z_i \in \mathbb{R}^{n_i+n_s} : \; h_{Z_j}(\text{proj}^{-1}(z_i)) \geq 0 \; \forall \, j = 1, ..., n_Z \right\} \tag{3.35}$$

for $i = 1, 2$. Recall that proj^{-1} is the back-projection operator from any subspace Z_i into Z as in (3.11). These definitions lead to the following lemma, which confirms that the projection and back-projection operators work "as expected" in mapping between Z and Z_i:

Lemma 21. *[CHT16, Section IV, Lemma 1] Let $z \in Z$, $z_i = \text{proj}_{Z_i}(z)$, and $S_i \subseteq Z_i$ for some subsystem, i. Then $z_i \in S_i \iff z \in \text{proj}^{-1}(S_i)$.*

Next, the definition of a self-contained subsystem is restated:

Definition 22. *[CHV+17, Definition 5] Consider the following special case of* (3.33)*:*

$$\begin{aligned} \dot{z}_1(t) &= f_1(t, z_1(t), z_s(t), k) \\ \dot{z}_2(t) &= f_2(t, z_2(t), z_s(t), k) \\ \dot{z}_s(t) &= f_s(t, z_s(t), k) \\ \dot{k}(t) &= 0. \end{aligned} \tag{3.36}$$

Each of the subsystems with states defined as $z_i = (z_i, z_s)$, for $i = 1,\ 2$, is called a self-contained subsystem *(SCS).* (3.36) *is called the* full system*.*

The SCS's in (3.36) show that the evolution of each subsystem depends only on the subsystem states: $\dot{z}_i$ depends only on $z_i = (z_i, z_s, k)$. Notice that the trajectory parameters k can appear in both SCS's. Given some initial condition $z_0 \in Z_0$, let, $z : [0, T] \to Z$ be a trajectory of the full system (3.36). Similarly, if $z_i : [0, T] \to Z_i$ is the trajectory of subsystem i, then z_i satisfies the following subsystem dynamics for all $t \in [0, T]$:

$$\begin{aligned} \dot{z}_i(t) &= \begin{bmatrix} f_i(t, z_i(t), z_s(t), k) \\ f_s(t, z_s(t), k) \end{bmatrix} \\ \dot{k}(t) &= 0. \end{aligned} \tag{3.37}$$

Trajectories of the full system are related to the trajectories of the subsystem via the projection operator, $\text{proj}_{Z_i}(z(t)) = z_i(t)$, from (3.9) [CHV+17, Equation (12)].

To account for tracking error, each error function (g_i from Assumption 7) must be defined independently for each subsystem, so that subsystems 1 and 2 are still SCS. The error function is added to (3.36) as defined below:

$$\begin{aligned} \dot{z}_1(t) &= f_1(t, z_1(t), z_s(t), k) + g_1(t, z_1(t), z_s(t), k) \circ d(t) \\ \dot{z}_2(t) &= f_2(t, z_2(t), z_s(t), k) + g_2(t, z_2(t), z_s(t), k) \circ d(t) \\ \dot{z}_s(t) &= f_s(t, z_s(t), k) + g_s(t, z_s(t), k) \circ d(t) \end{aligned} \tag{3.38}$$

The remainder of this section assumes subsystems 1 and 2, with states $z_1 = (z_1, z_s)$ and $z_2 = (z_2, z_s)$, have dynamics defined in (3.38). Subsystems 1 and 2 are SCS's, and Lemma 21 and (3.9) hold.

Example 23. *Recall the Rover's trajectory-producing model in* (3.30)*. Solving* (D^l) *for this model is memory intensive since the total dimension is* $n = 7$*, as discussed at the end of Section 3.3. However, this system can be decomposed into two separate SCS's:*

$$\dot{z_1}(t) = \begin{bmatrix} k_1 \cos(\theta(t)) - l_r \dot{\theta}(t,k) \sin(\theta(t)) \\ \dot{\theta}(t,k) \end{bmatrix} \tag{3.39}$$

$$\dot{z_2}(t) = \begin{bmatrix} k_1 \sin(\theta(t)) + l_r \dot{\theta}(t,k) \cos(\theta(t)) \\ \dot{\theta}(t,k) \end{bmatrix} \tag{3.40}$$

where $z_1 = [x, \theta]^\top \in Z_1$ *and* $z_2 = [y, \theta]^\top \in Z_2$*. The trajectory-tracking model* (3.13) *is produced for each SCS, by including error functions* g_1 *and* g_2 *as in Assumption 7.*

3.4.3 FRS Reconstruction

This section describes how to compute a time-varying FRS for each sub-system, then develop a reconstruction program to compute an overapproximation of $\mathcal{X}_{\text{FRS}}$ for the full system. Formally, the time-varying FRS is defined as:

$$\begin{aligned} \mathcal{Z}_{\text{TFRS}} = \Big\{ & (t,z,k) \in \ [0,T] \times Z \times K \mid \exists\ z_0 \in Z_0, \text{ and } d \in L_d \\ & \text{s.t. } z = \tilde{z}(t) \\ & \text{where } \dot{\tilde{z}}(\tau) = f(\tau, \tilde{z}(\tau), k) + g(\tau, \tilde{z}(\tau), k) \circ d(\tau) \\ & \text{a.e. } \tau \in [0,T] \text{ and } \tilde{z}(0) = z_0 \Big\}. \end{aligned} \tag{3.41}$$

An overapproximation of $\mathcal{X}_{\text{TFRS}}$ can be computed with a similar program and implementation to (D) in §3.3.2.

$$\inf_{v,w,q} \quad \int_{[0,T]\times Z\times K} w(t,z,k)\, d\lambda_{[0,T]\times Z\times K} \tag{D_T}$$

$$\begin{aligned} \text{s.t. } & \mathcal{L}_f v(t,z,k) + q(t,z,k) \leq 0, && (D_T1) \\ & \mathcal{L}_g v(t,z,k) + q(t,z,k) \geq 0, && (D_T2) \\ & -\mathcal{L}_g v(t,z,k) + q(t,z,k) \geq 0, && (D_T3) \\ & q(t,z,k) \geq 0, && (D_T4) \\ & -v(0,z,k) \geq 0, && (D_T5) \\ & w(t,z,k) \geq 0, && (D_T6) \\ & w(t,z,k) + v(t,z,k) - 1 \geq 0. && (D_T7) \end{aligned}$$

Constraints $(D_T1), (D_T2), (D_T3), (D_T4)$, and (D_T7) apply for all $(t, z, k) \in [0, T] \times Z \times K$. Constraint (D_T5) applies for all $(z, k) \in Z_0 \times K$. Constraint (D_T6) applies for all $(t, z, k) \in [0, T] \times Z \times K$. The given data in the problem are $f,\ g,\ Z, Z_0,\ K,$ and T. The infimum is taken over $(v, w, q) \in C^1([0, T] \times Z \times K) \times C([0, T] \times Z \times K) \times C([0, T] \times Z \times K)$. $\lambda_{[0,T]\times Z\times K}$ denotes the Lebesgue measure on $[0, T] \times Z \times K$.

Since subsystems 1 and 2 are SCS's, a time-varying FRS can be found for each separately using (D_T). Denote the two applications of (D_T) as $(D_T^{(1)})$ and $(D_T^{(2)})$, respectively. This section formulates an optimization program that overapproximates the intersection of the back-projections of the subsystems, thus overapproximating the FRS. This program is refered to as *reconstruction*. First define the intersection of the back-projections of v as:

$$\mathcal{V} = \{(z, k) \in Z \times K \mid \exists\, t \in [0, T] \text{ s.t. } v_1(t, z_1, k) \leq 0,\ v_2(t, z_2, k) \leq 0\}, \tag{3.42}$$

which is the set where both v_1 and v_2 are negative at any point in time. Next, let (v_1, w_1, q_1) (resp. (v_2, w_2, q_2)) be a feasible solution to $(D_T^{(1)})$ (resp. $(D_T^{(2)})$). An outer approximation of $\mathcal{X}_{\text{FRS}}$ can be reconstructed with the following optimization program:

$$\begin{aligned}
\inf_{w_r}\ & \int_{X\times K} w_r(x, k)\, d\lambda_{X\times K} && (R)\\
& w_r(x, k) \geq 1,\ \forall (x, k) \in \mathcal{V} && (R1)\\
& w_r(x, k) \geq 0,\ \forall (x, k) \in X \times K && (R2),
\end{aligned}$$

where $x = \text{proj}_X(z)$ and the infimum is taken over $w_r \in C(X \times K)$. Figure 3.6 shows the intersection of back-projections of the subsystem FRS's for the Rover. Now, I prove that the solution to (R) contains the FRS.

Theorem 24. *Let w_r be a feasible solution to (R). Then $\mathcal{X}_{\text{FRS}}$ is a subset of the 1-superlevel set of w_r.*

Proof. Let $z_0 \in Z_0$, $k \in K$, and $d \in L_d$ be arbitrary such that $z : [0, T] \to Z$ is a trajectory of the full system (3.36). Let $x = \text{proj}_X(z)$ and $X = \text{proj}_X(Z)$. Let $z_1(t) = \text{proj}_{Z_1}(z(t))$ give the corresponding trajectory of subsystem 1, and similarly let z_2 give the trajectory of subsystem 2. By Lemma 21, since the full system (3.36) is decomposable, $z(t) \in \text{proj}^{-1}(z_1(t)) \cap \text{proj}^{-1}(z_2(t))$. Recall that (v_1, w_1, q_1) is a feasible solution to $(D_T^{(1)})$, which denotes (D_T) solved with the dynamics of subsystem 1. By Lemma 15, $v_1(t, z_1(t), k)$ is non-positive and decreasing along the trajectory $z_1(t)$ for every $t \in [0, T]$, and similarly $v_2(t, z_2(t), k) \leq 0$ for $z_2(t)$. The set $\mathcal{V}$ (3.42) contains (x, k) in $X \times K$, such that $v_1(t, z_1, k) \leq 0$, $v_2(t, z_2, k) \leq 0$ and $t \in [0, T]$. Constraint $(R1)$ requires that $w_r(x, k) \geq 1$ if $(x, k) \in \mathcal{V}$. Since z_0, k, and d were arbitrary, the proof is

complete. □

3.4.4 Reconstruction Implementation

In this section, a relaxation of (R) is implemented with SOS polynomials and I show that the system decomposition method reduces the upper bounds on memory usage. Suppose $l \in \mathbb{N}$, and suppose (v_1^l, w_1^l, q_1^l) and (v_2^l, w_2^l, q_2^l) are feasible solutions to (D_{T1}^l) and (D_{T2}^l), which are (D_T^l) applied to Subsystems 1 and 2 respectively. Recall the sets H_T, H_Z, H_K, and H_X from Section 3.3.2, which contain the polynomials defining the sets $[0, T],\ Z,\ K$, and X respectively. Let $\alpha \in \mathbb{N}$ and $\alpha \geq l$. I pose the following SDP to reconstruct the FRS:

$$
\begin{aligned}
&\inf_{w_r^\alpha} \ y_{X\times K}^\top \text{vec}(w_r^\alpha) && && (R^\alpha)\\
&\quad w_r^\alpha - 1 && \in Q_{2\alpha}(-v_1, -v_2, H_T, H_Z, H_K) && (R1^\alpha)\\
&\quad w_r^\alpha && \in Q_{2\alpha}(H_X, H_K) && (R2^\alpha)
\end{aligned}
$$

where $x = \text{proj}_X(z)$ and $w_r^\alpha \in \mathbb{R}_{2\alpha}[x, k]$. As in (D^l), the vector $y_{X\times K}$ contains moments associated with the Lebesgue measure $\lambda_{X\times K}$, so $\int_{X\times K} w_r^\alpha(x,k) d\lambda_{X\times K} = y_{X\times K}^T \text{vec}(w_r^\alpha)$ for $w \in \mathbb{R}_{2\alpha[x,k]}$ [MVTT14].

The proposed system decomposition approach reduces memory usage since solving (D_T^l) for each subsystem reduces the problem dimension. In particular, the reconstruction program (R^α), only has two SOS constraints of degree 2α; hence, it has a less stringent memory requirement than (D^l). For the rover example, solving (D_T^4) for subsystems (3.39) and (3.40) each requires approximately 1.5×10^5 free variables and used 473 GB of RAM. The reconstruction program (R^5) requires approximately 5.5×10^4 free variables and used 227 GB of RAM. In contrast, recall from Section 3.3.3 that the RAM required for the full system with $l = 3$ was 504 GB. Figure 3.7 compares the FRS computed with the decomposed and reconstruction programs to an FRS computed for the full system (3.30) by solving (D^3).

With this section complete, one can compute a conservative approximation of the FRS for a wide class of mobile ground robots. Note, by Theorem 24, the 1-super level sets of w_r (and w_r^α) contain $\mathcal{X}_{\text{FRS}}$; therefore, subsequent theorems and lemmas in Chapter 4 pertaining to w (and w^l) also hold for w_r (and w_r^α).

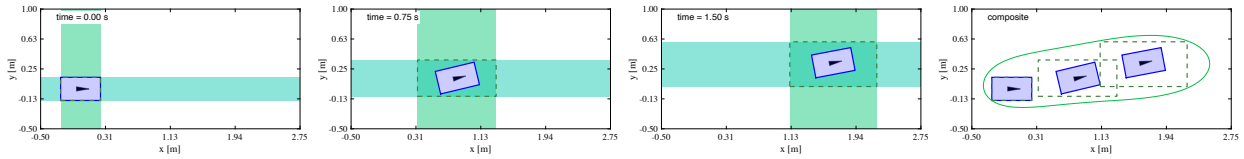
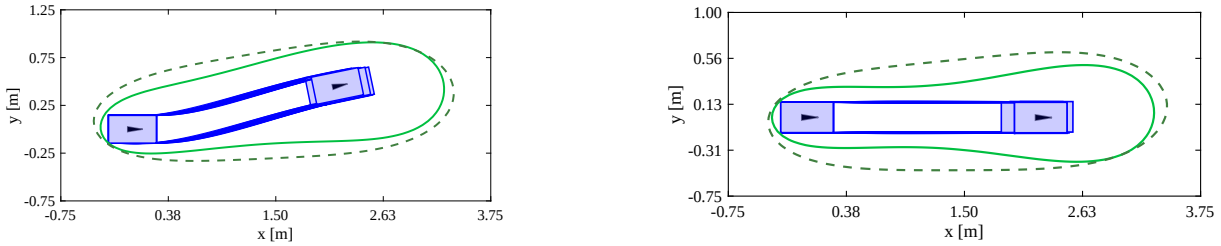


Figure 3.6: Example of the system decomposition and reconstruction for the FRS of the Rover's trajectory-producing system (3.30). The robot is the rectangle with a triangle indicating its heading. The FRS and robot at 0.0, 0.75, and 1.5 s following a trajectory with parameters $k = (1.1 \text{ m/s}, 0.5 \text{ rad/s}, 0.0 \text{ rad})$ are depicted from left to right. The vertical and horizontal bars show back-projections of the 0 sub-level sets of v_i^4 from $(D_T^{(i)})$ for $i = 1, 2$. The dashed rectangle indicates the intersection of the back projections. The far right figure shows the intersections at each time, along with the 1-level set of w_r^5 as a solid line.



(a) (b)

Figure 3.7: Comparison of reach sets computed for lane change trajectories produced by (3.30). The dark, dashed contours represent the 1-level set of w^3 computed for the full system. The light contours represent the 1-level set of w_r^5, computed using the system decomposition and reconstruction methods. The reach sets are computed with a time horizon of 1.5 s. Notice that the FRS computed with system decomposition is almost entirely contained within the FRS that does not use system decomposition; so, system decomposition reduces conservatism by enabling the computation of a higher-degree FRS. Example trajectories are generated by simulating the high-fidelity model for the rover. Subfigure (a) shows the trajectory parameter k = (2.0 m/s 0.5 rad/s, 0.0 rad). Subfigure (b) shows the trajectory parameter k = (1.6 m/s 0.0 rad/s, 0.0 rad).

3.5 Systems with Time-switching Dynamics

In receding-horizon motion planning, guaranteeing that a collision-free trajectory can be found at the next planning iteration is challenging, if not impossible, in arbitrary, dynamic environments. One school of thought is to plan a fail safe maneuver, that can be executed at the next planning iteration in the event a feasible solution to the trajectory optimization program cannot be found. For the ground robot case, braking to a stop is usually an appropriate fail-safe maneuver [SSSS17]. This motivates the computation of reachable sets for systems with time-switching dynamics that consist of a moving, braking, and stopped phase.

Example 25. *Consider the Segway in Example 4. Let τ_{plan} be the planning time and T be the time horizon for a planned trajectory. The following parameterization for the velocity,* $\text{vel} : T \times K \to \mathbb{R}$ *and yaw rate, $\dot{\theta} : T \times K \to \mathbb{R}$, produces a trajectory where the Segway moves for 1 planning iteration, then brakes to a stop.*

$$\begin{bmatrix} \text{vel}(t,k) \\ \dot{\theta}(t,k) \end{bmatrix} = \begin{cases} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix}, & t \in [0, \tau_{\text{plan}}], \\ \left(1 - \frac{t-\tau_{\text{plan}}}{\tau_{\text{brake}}}\right) \begin{bmatrix} k_1 \\ k_2 \end{bmatrix}, & t \in [\tau_{\text{plan}}, \tau_{\text{plan}} + \tau_{\text{brake}}] \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, & t \in [\tau_{\text{plan}} + \tau_{\text{brake}}, T]. \end{cases} \tag{3.43}$$

To compute a FRS for the system in Example 25, one can treat the system as a hybrid system [SVBT14] where the modes correspond to moving, braking, and stopping, and the guards are the times at which the dynamics switch. Unfortunately solving a SOS program for such a system For this example, the hybrid reachability analysis would increase the size of the FRS program roughly by a factor of 3. Recall that §3.3.3 discussed how memory intensive SOS programming is. To avoid the size increase, I developed a way to compute the FRSs separately for time-switching systems. First define the reachable set for the time switching system during phase $i \in \{1, \ldots, n_{\text{phases}}\}$:

$$\begin{aligned} \mathcal{Z}_{\text{TFRS},i} = \Big\{ & (t,z,k) \in [t_{i-1}, t_i] \times Z \times K \mid \exists\, z_0 \in Z_0, \text{ and } d \in L_d \\ & \text{s.t. } z = \tilde{z}(t), \text{ and for } j \in \{1, \ldots, i\} \\ & \dot{\tilde{z}}(\tau) = f_j(\tau, \tilde{z}(\tau), k) + g_j(\tau, \tilde{z}(\tau), k) \circ d(\tau), \text{ for } \tau \in [t_{j-1}, t_j] \\ & \text{a.e. } \tau \in [t_{j-1}, t_j] \text{ and } \tilde{z}(0) = z_0 \}, \end{aligned} \tag{3.44}$$

where in this application $t_0 = 0$ and $t_{n_{\text{phases}}} = T$. Minor modifications are made to (D_T) to compute

an FRS for the i^{th} phase as follows

$$\inf_{v_i, w_i, q_i} \int_{[t_{i-1}, t_i] \times Z \times K} w_i(t, z, k)\, d\lambda_{[t_{i-1}, t_i] \times Z \times K} \tag{D_{Ti}}$$

$$\text{s.t. } \mathcal{L}_{f_i} v_i(t, z, k) + q_i(t, z, k) \leq 0, \tag{$D_{Ti}1$}$$

$$\mathcal{L}_{g_i} v_i(t, z, k) + q_i(t, z, k) \geq 0, \tag{$D_{Ti}2$}$$

$$-\mathcal{L}_{g_i} v_i(t, z, k) + q_i(t, z, k) \geq 0, \tag{$D_{Ti}3$}$$

$$q_i(t, z, k) \geq 0, \tag{$D_{Ti}4$}$$

$$-v_i(t_{i-1}, z, k) \geq 0, \tag{$D_{Ti}5$}$$

$$w_i(t, z, k) \geq 0, \tag{$D_{Ti}6$}$$

$$w_i(t, z, k) + v_i(t, z, k) - 1 \geq 0. \tag{$D_{Ti}7$}$$

Constraints $(D_{Ti}1)$, $(D_{Ti}2)$, $(D_{Ti}3)$, $(D_{Ti}4)$, $(D_{Ti}6)$, and $(D_{Ti}7)$ apply for all $(t, z, k) \in [t_{i-1}, t_i] \times Z \times K$. Constraint $(D_{Ti}5)$ applies for all

$$(z, k) \in \begin{cases} Z_0 \times K, & i = 1 \\ \{(z, k) \in Z \times K \mid v_{i-1}(t_{i-1}, z, k) \leq 0\}, & \text{otherwise} \end{cases} \tag{3.45}$$

The given data in the problem are $f_i,\ g_i,\ Z, Z_0,\ K,$ and $[t_{i-1}, t_i]$ for $i \in \{1, \ldots, n_{\text{phases}}\}$. The infimum is taken over $(v, w, q) \in C^1([t_{i-1}, t_i] \times Z \times K) \times C([t_{i-1}, t_i] \times Z \times K) \times C([t_{i-1}, t_i] \times Z \times K)$. Note that this formulation requires solving (D_{Ti}) in increasing order of i, as the initial condition for each phase depends on the final reachable set for the prior phase. [SVBT14, Theorem 4] provides the following lemma:

Lemma 26. *[SVBT14, Theorem 4] Let (v_i, w_i, q_i) be a feasible solution to (D_{Ti}) for each phase $i \in \{1, \ldots, n_{\text{phases}}\}$. Let $\mathcal{Z}_{\text{TFRS},i}$ define the FRS at a point in time during phase i as in* (3.44) *For any $(t, z, k) \in \mathcal{Z}_{\text{TFRS},i}$, $v_i(t, z, k) \leq 0$.*

A sequence of SOS programs to solve each (D_{Ti}) can be implemented as described in §3.3.2, and the relevant theorems related to feasibility and convergence in Remark 19 apply to the time-switched system as well. An example of how the time-switching notation in (3.44) and (D_{Ti}) applies to the running example is given below.

Example 27. *Consider the Segway in Example 4 and the braking maneuver in Example 25. The dynamics have 3 phases with $i = \{1, 2, 3\}$ corresponding to $\{\text{moving}, \text{braking}, \text{stopped}\}$. The dynamics for each phase, f_i, are given by a Dubin's car model with velocity and yaw rates defined in order in* (3.43)*. Tracking error functions g_i are fit by applying Assumption 7 to each phase separately. The switching times are given by $t_0 = 0$, $t_1 = \tau_{\text{plan}}$, $t_2 = \tau_{\text{brake}} + \tau_{\text{plan}}$, and $t_3 = T$.*

For collision checking, one may wish to compute an FRS for n_I time intervals, possibly at a finer division than each phase. First, specify a collection of closed *time intervals* $\mathbf{t}_1, \ldots, \mathbf{t}_{n_I}$, $n_I \in \mathbb{N}$, that cover T, meaning $\cup_{j=1}^{n_I} \mathbf{t}_j = T$. Then each time interval can be written as $\mathbf{t}_j = t \in [t_j, t_{j+1}]$; note that $\mathbf{t}_1 = [0, t_1]$ and $\mathbf{t}_{n_I} = [t_{n_I-1}, T]$. Then the FRS for each interval is written as

$$\begin{aligned} \mathcal{X}_{\text{FRS},j} = \Big\{ (x,k) \in\ & X \times K \mid \exists\, t \in \mathbf{t}_j,\ z \in Z,\ i \in \{1, \ldots, N\} \\ & \text{s.t. } x = \text{proj}_X(z),\ (t,z,k) \in \mathcal{Z}_{\text{FRS},i} \}. \end{aligned} \tag{3.46}$$

Given a solutions for (D_{Ti}), one can solve the following program to compute $\mathcal{X}_{\text{FRS},j}$

$$\begin{aligned} \inf_{w} \quad & \int_{X \times K} w(x,k)\, d\lambda_{X \times K} && (D_I) \\ \text{s.t.} \quad & w(x,k) \geq 0, \text{ on } (x,k) \in X \times K && (D_I 1) \\ & w(x,k) - 1 \geq 0, \text{ on } \{(t,x,k) \in \mathbf{t}_j \times Z \times K \mid v_i(t,z,k) \leq 0\} && (D_I 2) \end{aligned}$$

Constraint $(D_I 2)$ holds for each $i \in \{1, \ldots, n_{\text{phases}}\}$. The infimum is taken over $w \in C(X \times K)$.

Theorem 28. *Let $(v_i, \tilde{w}_i, q_i)$ be feasible solutions to (D_{Ti}) for $i \in \{1, \ldots, n_{\text{phases}}\}$ and w be a feasible solution to (D_I). Then $(x,k) \in \mathcal{X}_{\text{FRS},j}$ implies $w(x,k) \geq 1$.*

Proof. Lemma 26 gives that v_i is negative along trajectories produced by the trajectory tracking model for $i \in \{1, \ldots, n_{\text{phases}}\}$. Constraint $(D_I 1)$ provides that w is greater than 1 when v is negative at any point in time $t \in \mathbf{t}_j$. □

Program (D_I) is implemented using the SOS polynomial procedure outlined in §3.3.2. Note that in practice one only needs to consider phases with switching times $t_i \in \mathbf{t}_j$ when formulating constraint $(D_I 2)$, since points with $t_i \notin \mathbf{t}_j$ are not contained in $\mathcal{Z}_{\text{FRS},i}$.

CHAPTER 4

Reachability-based Trajectory Design: Theory

This chapter gives an overview of Reachability-based Trajectory Design (RTD). RTD is an algorithm for provably safe, online trajectory optimization, using the reachable sets described in Chapter 3, §4.1 introduces Algorithm 1; the general Algorithm used for online planning. §4.2 describes two methods for representing static obstacles are represented as constraints in the trajectory optimization [KVJRV17, KVB+20]. §4.3 described two methods for representing dynamic obstacles as constraints in the trajectory optimization [VKL+19, VLK+19].

4.1 Online Planning

This section describes the receding-horizon algorithm (Algorithm 1) used for online planning. In §4.1.1 the algorithm is summarized and a general form of the optimization problem over the space of trajectory parameters is given (see §3.2 for an explanation of trajectory parameters). In §4.1.2 sensing, and consequentially run time, limits to ensure the robot never has an at-fault collision are defined.

4.1.1 Trajectory Optimization

This section presents assumptions about environment, the general trajectory optimization problem, and the methodology used for online planning, Algorithm 1. Before discussing the environment, note that Algorithm 1 is a receding-horizon planner, meaning it plans and begins executing a trajectory of length T, every τ_{plan} s, where $T > \tau_{\text{plan}}$.

The robot's environment and assumptions about the information that it receives are now described. Obstacles, the primary constraints in the trajectory optimization problem, are defined as follows:

Definition 29. *At any time $t \geq 0$, an* obstacle *is a subset of X that the robot cannot intersect without being in collision (e.g., physical objects or other actors). Denote the i^{th} obstacle at t by*

$O_t^i \subset X$ *for each* $i \in \{1, \ldots, n_{\text{obs}}\}$.

The ability to sense and predict obstacles and their motion is required.

Assumption 30. *The robot senses all obstacles within a distance* $D_{\text{sense}} > 0$ *of the robot's footprint, and predicts their behavior at least* $\tau_{\text{plan}} + T$ *seconds into the future.*

D_{sense} is called the *sensor horizon*. Predictions are discussed next.

Definition 31. *A* conservative prediction *is a map* $P : \mathbb{R}_{\geq 0} \to \mathcal{P}(X)$ *that contains all obstacles within* D_{sense} *(Assumption 30) at each time* $t \geq 0$*; i.e.,* $P(t) \supseteq \bigcup_i O_t^i$.

By 31, each obstacle is contained within its prediction, meaning a negative collision checking against the prediction, results in a negative collision check against the obstacle. Creating predictions that satisfy Assumption 30 and Definition 31 is the topic of open research, but is not the focus of this Chapter. The focus of this chapter is provide tools for motion planners to incorporate such predictions in a provably safe manner, for example the developed approach will be amenable to using zonotope-based predictions from [KA17]. To set a lower bound on the sensor horizon, assume the following:

Assumption 32. *There are up to* $n_{\text{obs,max}}$ *obstacles sensed at any time; i.e.,* $n_{\text{obs}} \leq n_{\text{obs,max}}$. *The speed of all obstacles is bounded by* $v_{\text{obs,max}} \geq 0$.

Occluded regions can be treated as dynamic obstacles [YVJR19] that can be conservatively predicted as moving at $v_{\text{obs,max}}$ in any direction, or can be subject to specific rules.

The trajectory optimization problem to find optimal the trajectory parameters $k \in K$ for the robot to execute is now described. See §3.2 for a description and examples of the trajectory parameters. One wishes to optimize over the safe set of trajectory parameters, defined as

Definition 33. *The safe set of trajectory parameters* K_{safe} *is the set of parameters such that the robot, executing trajectory* k*, will not intersect a prediction of an obstacle at any point in the time horizon, i.e.*

$$K_{\text{safe}} = \{k \in K \mid \text{ego robot at } z_{\text{hi}}(t; z_{\text{hi},0}, k) \cap P(t) = \emptyset, \forall\, t \in [0, T]\}, \tag{4.1}$$

where $z_{\text{hi}}(t; z_{\text{hi},0}, k)$ *is the solution of* (3.4) *at time* t *with initial condition* $z_{\text{hi},0} \in Z_{\text{hi},0}$ *and controller* u_k.

Given a cost function $J : K \to \mathbb{R}$, that attempts to accomplish a goal specified by the high level planner, the general trajectory optimization program is

$$\begin{aligned} &\min_k J(k) \\ &\text{s.t. } k \in K_{\text{safe}} \end{aligned} \tag{4.2}$$

The major contribution in this chapter is to use the reachable sets from Chapter 3 to write down the constraint in (4.2) as a set of inequality functions $\Phi : K \to \mathbb{R}^{n_{\text{cons}}}$, i.e.

$$\Phi(k) \leq 0 \implies k \in K_{\text{safe}}, \tag{4.3}$$

where n_{cons} is finite. §4.2 and §4.3 explain how to construct Φ, given predictions for static and dynamic obstacles, for now assume this is handled by a function called `generateConstraints`. The trajectory optimization can then be written in standard form as a nonlinear program:

$$\begin{aligned} &\min_{k \in K} J(k) \\ &\text{s.t. } \Phi(k) \leq 0. \end{aligned} \tag{4.4}$$

Remark 34. *In the subsequent sections, strict inequalities, $\Phi(k) < 0$, will be enforced. Strict inequalities can be transformed into non-strict inequalities, as in* (4.3)*, by adding a small positive value, $\epsilon > 0$, to the left side, i.e. $\Phi(k) + \epsilon \leq 0$.*

Trajectory optimization is solved in a receding-horizon fashion using Algorithm 1. Algorithm

Algorithm 1 RTD Online Planning

1: **Require:** $k_0 \in K$, $z_{\text{hi},0}$, and cost function $J : K \to \mathbb{R}$.
2: **Initialize:** $j = 0$, $t_j = 0$, $k^* = k_0$, $z_{\text{hi},j} = z_{\text{hi},0}$.
3: **Loop:** // Line 4 executes at the same time as Lines 5–9
4: **Execute** k^* for $[t_j, t_j + \tau_{\text{plan}})$
5: $P \leftarrow$ `senseAndPredictObstacles()`.
6: $\Phi \leftarrow$ `generateConstraints`(P).
7: **Try** $k^* \leftarrow \text{argmin}_k\{J(k) \mid \Phi(k) \leq 0,\ k \in K\}$ for duration τ_{plan}
8: **Catch** continue // k^* is unchanged
9: $z_{\text{hi},j+1} \leftarrow$ `estimateFutureState`$(t_j + \tau_{\text{plan}}, z_{\text{hi},j+1}, k^*)$
10: $t_{j+1} \leftarrow t_j + \tau_{\text{plan}}$ and $j \leftarrow j + 1$
11: **End**

1 takes in an initial trajectory parameter k_0, initial state of the robot $z_{\text{hi},0}$, and a user-specified cost function $J : K \to \mathbb{R}$ that attempts to accomplish a goal given by the high level planner. The algorithm begins by executing the initial trajectory parameter for τ_{plan} s in Line 4. Lines 5 and 6 sense obstacles, generate predictions, then generate constraints as in (4.3). Line 7 attempts to solve (4.4) in τ_{plan} s. If successful Line 7 returns a new trajectory parameter k^* to execute at the next planning iteration; however there is no guarantee that (4.4) has a feasible solution or can be solved within τ_{plan} s. If unsuccessful, the robot continues to execute the current plan (section §4.1.2 discusses how to design k to ensure this is safe). Line 9 predicts the state of the robot for the next planning iteration using the high-fidelity model 3.4. On hardware implementations, error

in the state prediction in Line 9 must be accounted for in the FRS computation, as discussed in Assumption 11. Line 10 updates the time and counters to prepare for the next iteration.

4.1.2 Fault and Sensing Limits

This section discusses the notion of fault and sensing limits required to ensure that a robot executing Algorithm 1 is always safe. In environments with dynamic obstacles it may be impossible to guarantee collision avoidance; but one can at least guarantee that will not be at-fault for a collision as defined below:

Definition 35. *Let $t \geq 0$ be the current time. If robot is moving at time t, it is* not-at-fault *if not intersecting any obstacle O_t^i. If the robot is stationary at time t, it is always* not-at-fault.

A more complicated definition of fault could also be considered, such as one that requires giving surrounding actors enough space to brake to a stop or safely swerve away from the ego robot. This would require placing assumptions on how surrounding vehicles or agents respond to the ego robot's motion (e.g. reaction time or rationality) [SSSS17].

Definition 35, coupled with 31, and Algorithm 1 Line 8 motivates the idea of planning *fail-safe* maneuvers. The idea of fail-safe maneuvers is to plan trajectories that end in a not-at-fault state, so that, if the robot cannot find a feasible trajectory in the next planning iteration, it can always execute the fail safe computed at the previous one. Similar strategies have been adopted in the literature [KTF+09, PA18]. I make the following remark about fail-safe maneuvers for this work.

Remark 36. *In this work, in accordance with Definition 35, the fail-safe maneuver is braking to a stop; then the robot stays stopped until a new input is found. In practice, this means to guarantee not-at-fault planning the trajectory-producing model* (3.2) *used in the reachable set computation, will end with a braking maneuver (see Example 25).*

Recall that the robot has to plan using the predictions available at the beginning of each planning iteration. So, it must be able to sense obstacles that could cause a collision while it tracks a desired trajectory that begins at the end of each planning iteration. This means there must be a lower bound on the robot's sensor horizon (i.e., the robot must detect obstacles from sufficiently far away). First assume the existence of a bounding function for ego robot's motion:

Assumption 37. *There is a function $D_{\max} : [0, T] \to \mathbb{R}_{\geq 0}$ that bounds the maximum distance the robot can travel over the planning horizon*

$$\max_{k \in K} \|\text{proj}_X(z_{\text{hi}}(t; z_{\text{hi},0}, k) - z_{\text{hi},0})\|_2 \leq D_{\max}(t) \tag{4.5}$$

where $z_{\text{hi}}(t; z_{\text{hi},0}, k)$ indicates trajectories of (3.4) at time t, starting from $z_{\text{hi},0} \in Z_{\text{hi},0}$ with controller u_k. This assumption is valid because of Assumptions 2 and 3 and that T is finite.

Recall that obstacles have a maximum speed $v_{\text{obs,max}}$ by Definition 29. The minimum sensor horizon required to guarantee not-at-fault planning with Algorithm 1 is specified in the following theorem.

Theorem 38. *Let the current time be 0 WLOG, D_{max} be given as in Assumption 37, and suppose the robot is currently executing a trajectory that is not-at-fault for $t \in [0, T]$ (Algorithm 1 Line 4) while planning with Algorithm 1, with fail-safe maneuvers as in Remark 36. If the robot's sensor horizon is*

$$D_{\text{sense}} \geq D_{\text{max}}(T) + D_{\text{max}}(\tau_{\text{plan}}) + (T + \tau_{\text{plan}})v_{\text{obs,max}}, \tag{4.6}$$

Then, no obstacle whose points all lie farther than D_{sense} from the robot at the current time can cause an at-fault collision with the robot at any $t' \in [0, \tau_{\text{plan}} + T]$.

Proof. The proof focuses on the interval $[0, \tau_{\text{plan}} + T]$, because as the robot is planning a trajectory to begin executing at τ_{plan}; while the ego robot executes the current desired trajectory, only obstacles within a distance $d_1 = D_{\text{max}}(\tau_{\text{plan}}) + \tau_{\text{plan}} v_{\text{obs,max}}$ could cause a collision. There are 2 possible scenarios. First the robot finds a new feasible trajectory (Algorithm 1 Line 7 is successful). Then, only obstacles within at least $d_2 = D_{\text{max}}(T) + T v_{\text{obs,max}}$ of the robot at time $t = \tau_{\text{plan}}$ could cause a collision when the robot begins tracking the new desired trajectory. In this case D_{sense} must be greater than $d_1 + d_2$, which it is by (4.6). In the second scenario, the robot continues executing the previous plan, which ends in a braking maneuver as per Remark 36. Then, obstacles within at least d_2 of the robot at time $t = 0$ could cause a collision. Since $D_{\text{sense}} > d_2$, the proof is complete. □

Note that the minimum sensor horizon (4.6) is a function of the length of the planned trajectory T, replanning rate τ_{plan}, and speed of the robot and obstacles. Since D_{sense} is also limited by hardware capabilities, it is important that $\tau_{\text{plan}} + T$ is low, especially when the speed of the robot and obstacles is high. An example of an autonomous vehicle operating on a highway illustrates this relationship

Example 39. *Suppose an autonomous vehicle driving in a straight line has a trajectory parameterization where it moves at a commanded speed v for $t \in [0, \tau_{\text{plan}}]$ then decelerates to a stop at 8 m/s^2. Suppose the perception module is able to sense obstacles within $D_{\text{sense}} = 100$ m. Assume the ego vehicle has a maximum speed of 25 m/s, so $D_{\text{max}}(t) = 25t$. Assume obstacles are static,*

$v_{\text{obs,max}} = 0$. *The planning time* τ_{plan} *must be*

$$100 \geq 25(T) + 25(\tau_{\text{plan}}) \tag{4.7}$$

$$100 \geq 25\left(\frac{25}{8} + 2\tau_{\text{plan}}\right) \tag{4.8}$$

$$0.4375 \geq \tau_{\text{plan}} \tag{4.9}$$

The bound in Theorem 38 assumes the worst-case motion for the obstacle. In environments with dynamic obstacle, it may be useful to consider cases that consider a bound on the relative velocity between the ego robot and other obstacles. For driving tasks such as in Example 39, the static obstacle case is equivalent to an assumption that other vehicles will not drive towards the ego vehicle and cause a head-on collision.

4.2 Static Obstacles

This section discusses how to formulate constraints for the trajectory optimization program for static obstacles. Two types of obstacle representations are covered. Semi-algebraic sets in S4.2.1 from [KVJRV17], and Discrete points in S4.2.2 from [KVB$^+$20]. In this section assume (D) has been solved with Sums-of-Squares programming as in §3.3.2, so a polynomial $w : X \times K \to \mathbb{R}$ whose 1 superlevel set contains the forward reachable set of the ego robot at all timepoints $t \in [0, T]$ (see $\mathcal{X}_{\text{FRS}}$ (3.18)) exists. Also assume that obstacles are static so $P(t_1) \equiv P(t_2)$ for any $t_1, t_2 \in [0, T]$; in this section P is written without time as an argument.

4.2.1 Semi-algebraic Sets

In this section assume predictions are represented as semi-algebraic sets

Assumption 40. *The prediction,* P*, of static obstacles is represented by a collection of semi-algebraic sets, defined in the space of position* X*.*

$$P = \{x \in X \mid h_{P_i}(x) \geq 0, \text{ for any } i \in \{1, \ldots, n_{\text{obs}}\}\} \tag{4.10}$$

where h_P *are polynomial functions.*

The outerapproximation of the $\mathcal{X}_{\text{FRS}}$, given by the 1 superlevel set of w from the SOS implementation of (D), is intersected with the obstacles to generate an innerapproximation of the set of

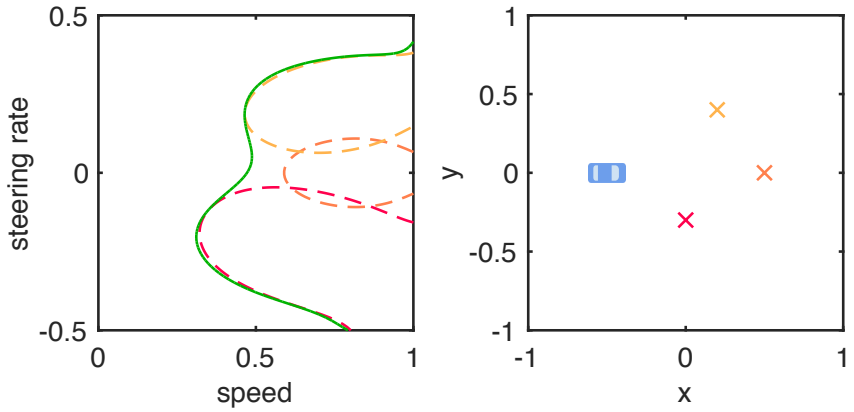


Figure 4.1: An example of set intersection, with P chosen as three points in X. On the right is the (x, y) subspace of X with each point obstacle shown, and the vehicle plotted in blue. On the left is the trajectory parameter space K, with three dashed-line contours containing an outer approximation of the trajectory parameters that would cause a collision with each point shown on the right (the colors match between the points and contours). The 0 level set of Φ returned from the set intersection program (4.11) is shown by the green contour (with the sub-level set to the left), which outer approximates all trajectory parameters that could result in collisions with any of the three points in P. Therefore, K_{safe} is inner-approximated.

safe trajectory parameters with the following *set intersection* program

$$\begin{aligned} \inf_{\Phi} \quad & \int_K \Phi(k)\, d\lambda_K \\ \text{s.t.} \quad & \Phi(k) \geq 0, \text{ on } \{(x,k) \in P \times K \mid w(x,k) \geq 1\}, \\ & \Phi(k) + 1 \geq 0, \text{ on } K, \end{aligned} \tag{4.11}$$

where w is the given data and the infimum is taken over $\Phi \in C(K)$. Since, by Assumption 40 all of the functions in (4.11) are polynomial and it can implemented with SOS programming.

Theorem 41. *Let w be a feasible function to (D), and $\Phi : K \to \mathbb{R}$ be a feasible function to* (4.11)*. Then $\{k \in K \mid \Phi(k) < 0\} \subset K_{\text{safe}}$.*

Proof. I will give a proof by contradiction. Let $z_{\text{hi}}(t; z_{\text{hi},0}, k)$ be a trajectory of the robot (3.4) at time t, with initial condition $z_{\text{hi},0}$, controller u_k that intersects P, and $\Phi(k) < 0$. By Lemmas 9 and 16, $w(\text{proj}_X(z_{\text{hi}}(t; z_{\text{hi},0}, k)), k) \geq 1$ at any $t \in [0, T]$. The first constraint in (4.11) gives that $\Phi(k) \geq 0$ for any $(x, k) \in P \times K$ where $w(x, k) \geq 1$, a contradiction. □

As a result of Theorem 41, one can use $\Phi(k) < 0$, as an inequality constraint in (4.4) to guarantee collision avoidance. Figure 4.1 shows an example of Φ returned from the set intersection program (4.11) for 3 point obstacles. The vehicle dynamic model is the Segway from Example 4. Figure 4.2

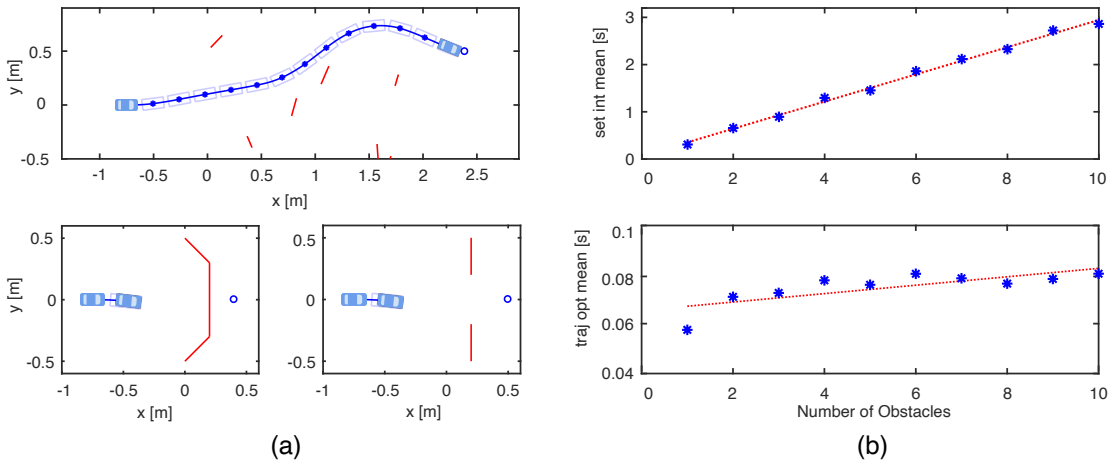


Figure 4.2: The top plot in subplot (a) shows an example result where vehicle begins on the left and reaches a randomly-generated goal, plotted as a blue circle. Every $\tau_{\text{plan}} = 0.5$ s, the vehicle replans its trajectory, shown by an asterisk plotted on the global trajectory in blue. In the bottom-left subplot, an obstacle was constructed to force an emergency braking maneuver. In the bottom-right subplot, an obstacle was constructed with a hole, but the FRS is overly conservative, resulting in a braking maneuver. Subplot (b) shows the mean set intersection time (4.11, top) and trajectory optimization time (4.4, bottom) versus the number of obstacles. Set intersection takes up to 3 s, and scales linearly with the number of obstacles. Trajectory optimization takes around 80 ms and has low correlation with number of obstacles.

(a) shows the Segway model navigating fields of line obstacles by solving the trajectory optimization program (4.4) with Φ from (4.11). 1000 trials with randomly placed obstacles were conducted and no crashes were observed. There are some gaps the Segway is not able to pass through due to conservatism of the FRS. Figure 4.2 (b) shows timing results of (4.11) and (4.4) for 1000 random trials with varying numbers of obstacles. (4.4) is able to solve within 0.1 s with a low correlation on the number of obstacles; however (4.4) takes up to 3 s to solve, and increases linearly with the number of obstacles.

4.2.2 Discrete Points

This section presents a method of representing the robot's environment as a finite number of discrete points. Representing the obstacles as discrete points means the inequality constraints Φ, end up being a list of polynomial constraints, where w, from (D), is evaluated at each obstacle point,

i.e. This leads to a finite number of inequality constraints for (4.4):

$$\Phi(k) = \begin{bmatrix} w(p_1, k) - 1 \leq 0 \\ w(p_2, k) - 1 \leq 0 \\ \vdots \\ w(p_n, k) - 1 \leq 0 \end{bmatrix} \implies k \in K_{\text{safe}} \tag{4.12}$$

where $X_p = \bigcup_{i=1}^{n} p_i$ is a set of discrete points in X. This method eliminates the set intersection intersection program from the prior section, (4.11), and replaces it with computations to generate X_p. The results Chapter 5 show this enables real time operation of Algorithm 1.

This section starts with preliminary assumptions in §4.2.2.1. §4.2.2.2 defines and motivates important quantities for buffering obstacles. §4.2.2.3 gives examples of the quantities for buffering rectangular and circular robots. §4.2.2.4 summaries Algorithm 2, which constructs X_p. A more thorough treatment for arbitrary convex shapes can be found in Appendix B.

4.2.2.1 Preliminary Assumptions

This section makes necessary assumptions about the robot footprint and obstacles.

Assumption 42. *The robot's footprint $X_0 \subset X$ is compact and convex with nonzero volume.*

Footprints fulfilling this assumption, such as circles and rectangles, are common for ground robots (consider the Segway and Rover in Figure 3.2).

The following general representation is used for predictions:

Assumption 43. *Each prediction $P_i \in P$ is a closed polygon with a finite number of vertices and edges.*

Note that these polygons are not necessarily convex. This assumption holds for common obstacle representations such as occupancy grids or line segments fit to planar point clouds. If an obstacle is not a closed polygon within the sensor horizon (such as a long wall), it can be closed by intersection with the sensor horizon D_{sense} (as in Assumption 30), which can be over-approximated by a regular polygon (the intersection is a closed set [Mun00, Theorem 17.1]). Note that P may contain one or more obstacles; the definitions and proofs in this section still hold if P is a union of polygons, which is itself a (potentially disjoint) polygon [FHW12]. Therefore, P is referred to as a singular obstacle for ease of exposition.

A projection mapping is now defined to to help navigate between state space and the space of parameters. Suppose that the tuple (v, w, q) is an optimal solution to Program (D) from Section

3.3.1. Then by Lemma 16, $w : X \times K \to \mathbb{R}$ over approximates an indicator function on $\mathcal{X}_{\text{FRS}}$. Define the set-valued maps $\pi_K : \mathcal{P}(X) \to \mathcal{P}(K)$ and $\pi_X : \mathcal{P}(K) \to \mathcal{P}(X)$ as

$$\pi_K(X') = \{k \in K \mid \exists\, x \in X' \text{ s.t. } w(x,k) \geq 1\}, \tag{4.13}$$

$$\pi_X(K') = \{x \in X \mid \exists\, k \in K' \text{ s.t. } w(x,k) \geq 1\}. \tag{4.14}$$

π_K is called the FRS *parameter projection map*. If $X' \subset X$, $\pi_K(X')$ are the parameters that could cause a collision with X'. The word "projection" is used for these operators to relate them to the projection operators proj_{Z_i} in Definition 6. Similarly, π_X and π_K return points in a subspace of the reachable set $\mathcal{X}_{\text{FRS}}$ that are identified by the function w. The utility of this map as it relates to point obstacles is demonstrated with the following lemma.

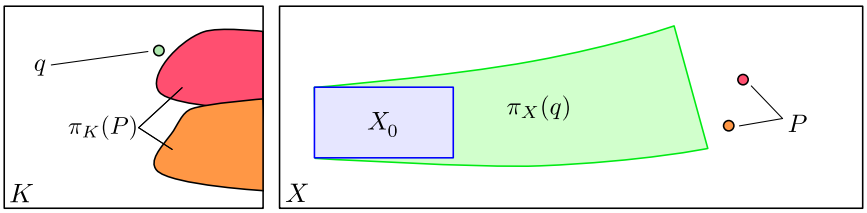
Lemma 44. *Suppose that the tuple (v, w, q) is an optimal solution to Program (D), and π_K is defined as in* (4.13)*. Consider an arbitrary point $p \in X \setminus X_0$. Let $k \in \pi_K(p)^C$. At $t = 0$, let the robot, described by the high-fidelity model* (3.4)*, be at the state $z_{\text{hi},0} \in Z_{\text{hi}}$. Suppose the robot tracks the trajectory parameterized by k, producing the high-fidelity model trajectory $z_{\text{hi}} : [0, T] \to Z_{\text{hi}}$. Then, no point on the robot's body ever reaches p. More precisely, there does not exist any pair $(t, z_{\text{hi},0}) \in [0, T] \times Z_{\text{hi},0}$ such that $p = \text{proj}_X(z_{\text{hi}}(t))$.*

Proof. Suppose for the sake of contradiction that there exists some $t \in [0, T]$ and $z_{\text{hi},0} \in Z_{\text{hi},0}$ for which $p = \text{proj}_X(z_{\text{hi}}(t))$. By Lemma 9, there exists $d \in L_d$ such that the trajectory-tracking model (3.13) has a trajectory $z : [0, T] \to Z$ for which $\text{proj}_Z(z_{\text{hi}}(t)) = z(t)$ at t. Then, $w(p, k) \geq 1$ by Lemma 16. But, by (4.13), $k \in \pi_K(p)^C$ implies that $w(p, k) < 1$, which is a contradiction. □

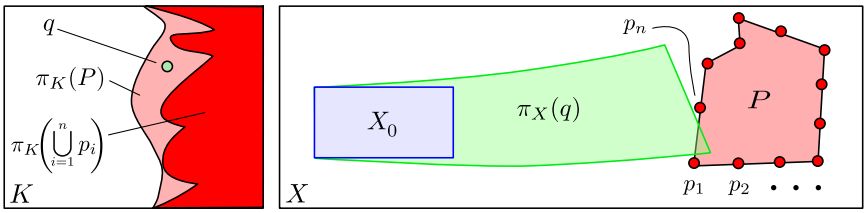
4.2.2.2 Buffer and Point Spacing Motivation

The goal of this section is to construct the discretized obstacle, $X_p \subset X$, such that if the robot cannot collide with the discretized points, it cannot collide with the obstacle. To that end, consider constructing X_p from points on the boundary of P, as illustrated in Figure 4.3b. Since the high-fidelity model of the robot (3.1) produces continuous trajectories in the subspace X (see Assumption 2), the robot cannot collide with an obstacle without passing through the obstacle's boundary.

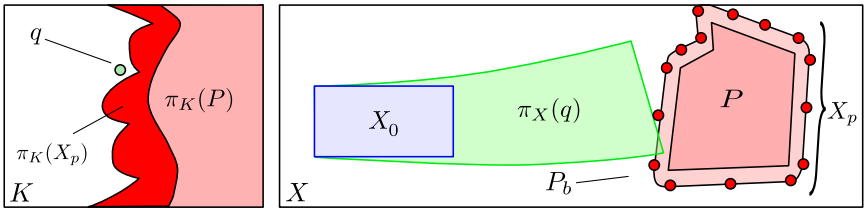
However, constructing X_p with a finite number of points on ∂P may be insufficient to prevent collisions. To see why, consider a candidate discretized obstacle $X_p = \{p_1, p_2, \ldots, p_n\} \subset \partial P$, with $n \in \mathbb{N}$. Then any $k \in \pi_K(X_p)$ may cause the robot to reach one or more $p_i \in X_p$. Suppose $q \in \pi_K(X_p)^C$. There is no guarantee that $\pi_X(q) \cap P = \emptyset$, i.e. that q would not cause a collision with the obstacle, because the robot may be able to travel between adjacent points in X_p as shown



(a)



(b)



(c)

Figure 4.3: Motivation and method for buffering and discretizing predictions. The robot has footprint X_0 in the xy-subspace X on the right, and the trajectory parameter space K is on the left. In Figure 4.3a, the P consists of two points, to illustrate the map π_K, which maps each point to a subset of K containing all trajectory parameters that could cause the robot to reach either point; since $q \in \pi_K(P)^C$, by Lemma 44, the robot cannot collide with either obstacle point. Figure 4.3b shows an arbitrary polygonal prediction (as in Assumption 43) with a set of discrete points $\{p_1, \ldots, p_n\}$ sampled from its boundary. These points are mapped to the subset of the parameter space K labeled $\pi_K\left(\bigcup_{i=1}^{n} p_i\right)$. A parameter q is chosen outside of the parameters corresponding to these points, but still lies within the projection of the actual obstacle $\pi_K(X_{\text{obs}})$, and therefore may cause a collision as illustrated by the set $\pi_X(q)$. Figure 4.3c shows the same obstacle, but buffered. The boundary of the buffered obstacle is sampled to produce the discrete, finite set X_p. The trajectory parameters corresponding to X_p are a superset of the unsafe parameters $\pi_K(X_{\text{obs}})$, so the robot cannot collide with the obstacle despite the FRS spatial projection $\pi_X(q)$ penetrating between two of the points of X_p.

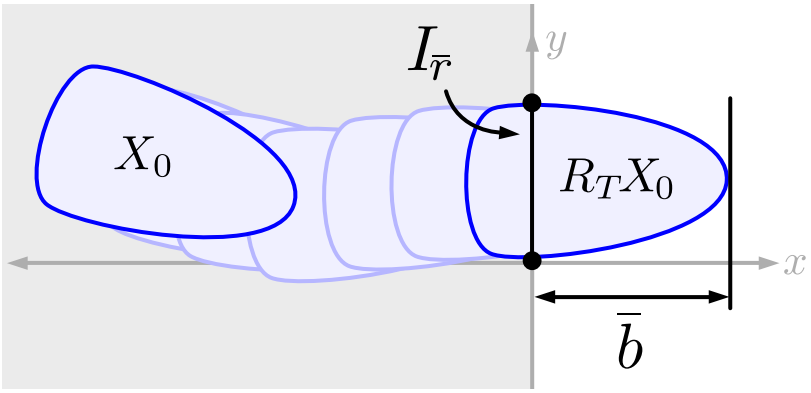


Figure 4.4: Maximum penetration distance $\bar{b}$, as defined in Lemma 46. X_0 is the robot's footprint, translated and rotated by R_T, and $I_{\bar{r}} \subset (X \setminus X_0)$ is a line segment of length $\bar{r}$.

in Figure 4.3b. To address this issue, the obstacle is buffered, then points from its boundary are selected with a maximum point spacing allowed between the points.

The purpose of this section, then, is to define the *buffer* and *point spacing* to enable constructing X_p.

Definition 45. *Let $b > 0$ be a distance, called a* buffer. *The* buffered prediction, *$P_b \supset P$, is a compact subset of X such that the maximum Euclidean distance between P and P_b is b:*

$$P_b = \{p \in X \mid \exists\, p' \in P \text{ s.t. } \|p - p'\|_2 \leq b\}\,. \tag{4.15}$$

Buffering an obstacle by b reduces the amount of free space available for the robot to navigate through. One can find an upper bound $\bar{b}$ on b, that gives the maximum distance the robot's footprint can penetrate

Lemma 46. *Let X_0 be the robot's footprint at time 0 (as in Definition 12), with width $\bar{r}$ (as in Definition 49). Let $I_{\bar{r}} \subset (X \setminus X_0)$ be a line segment of length $\bar{r}$. Then there exists a* maximum penetration distance *$\bar{b}$ that can be achieved by passing X_0 through $I_{\bar{r}}$.*

Figure 4.4 provides an example of the quantity $\bar{b}$ as defined in Lemma 46.

Having established the buffer b and its upper bound $\bar{b}$, the point spacing can be defined. First, the following lemma describes the geometry of the buffered obstacle.

Lemma 47. *The boundary of the buffered obstacle, consists of a finite set of line segments L and a finite set of arcs A of radius b. More precisely, let $n_L \in \mathbb{N}$ (resp. $n_A \in \mathbb{N}$) denote the number of line segments (resp. arcs). Let $L_i \in L$ (resp. $A_i \in A$) denote the i^{th} line segment (resp. arc). Note that each L_i and A_i is a subset of X. Then the boundary of the buffered obstacle can be written as*

the union of all of the lines and arcs:

$$\partial P_b \quad = \quad \left(\bigcup_{i=1}^{n_L} L_i \right) \ \cup \ \left(\bigcup_{i=1}^{n_A} A_i \right). \tag{4.16}$$

Now, consider a discretized obstacle X_p that is generated by selecting a set of points from ∂P_b such that the points are spaced by a distance $r > 0$ along the line segments and by a distance $a > 0$ along the arcs, as illustrated in Figure 4.3c.

Definition 48. $r > 0$ *is called the* point spacing *and* $a > 0$ *the* arc point spacing.

Section 4.2.2.3 shows that, by selecting r and a as function of the buffer b, the robot cannot pass completely between any pair of points in X_p and collide with an obstacle. Similar to the upper bound $\bar{b}$ on the buffer, one can find an upper bound $\bar{r}$ for r.

Definition 49. *The quantity* $\bar{r}$ *denotes the* maximum point spacing*, which is equal to the width of the robot footprint* X_0.

Recall that $\bar{b}$ limits the buffer b, to prevent obstacles from taking up too much free space. On the other hand, $\bar{r}$ makes sure that the point spacing is small enough that the discretized obstacle can be used to ensure safety; that is, the points in the X_p must be close enough that the robot cannot pass between them. r itself is used as an upper bound of a. Now the geometric quantities b, $\bar{b}$, r, a, and $\bar{r}$ are motivated Examples for common robot footprints are given in the next section.

4.2.2.3 Examples for Common Footprints

This section provides examples of $\bar{r}$, $\bar{b}$, r, and a for rectangular and circular robot footprints. For rigorous proofs of how to derive these quantities for arbitrary convex robot footprints, refer to Appendix B.

Example 50. *(Rectangular footprint). Suppose the robot footprint,* X_0*, is a rectangle of length* L *and width* W*, with* $L > W$*. Then* $\bar{r} = W$ *and* $\bar{b} = W/2$*. Given* $b \in (0, \bar{b})$*,* $r = 2b$ *and* $a = 2b\sin(\pi/4)$*. A visual proof is in Figure 4.5a.*

Example 51. *(Circular footprint). Suppose the robot footprint* X_0 *is a circle of radius* R*. Then* $\bar{r} = 2R$ *and* $\bar{b} = R$*. Pick* $b \in (0, \bar{b})$*. Define the positive angles* $\theta_1 = \cos^{-1}\left(\frac{R-b}{R}\right)$ *and* $\theta_2 = \cos^{-1}\left(\frac{b}{2R}\right)$*. Then set* $r = 2R\sin\theta_1$ *and* $a = 2b\sin\theta_2$*. A visual proof is in Figure 4.5b.*

This completes finding the geometric quantities $\bar{r}$, $\bar{b}$, r, and a. The next section uses these quantities to construct the discretized prediction, and proves that this enables identifying the set of safe trajectory parameters K_{safe}.

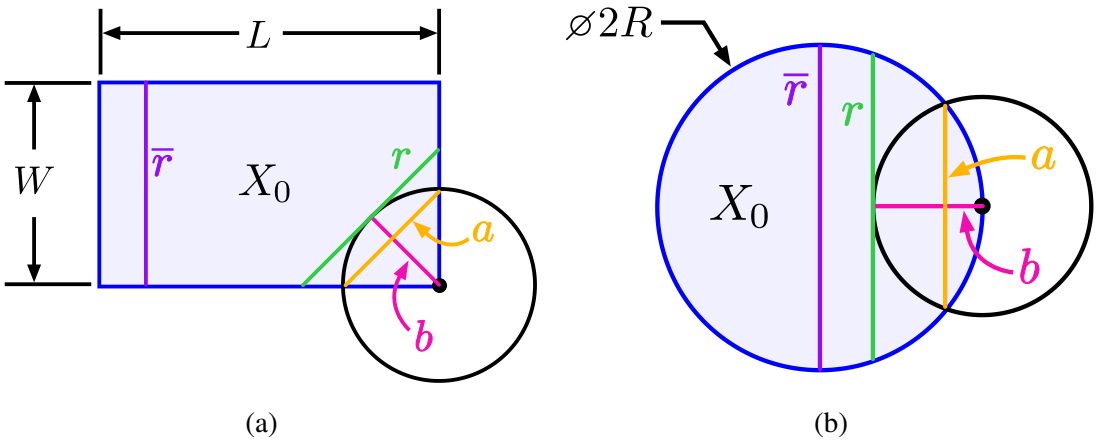


Figure 4.5: An illustration of the numbers $\bar{r}$, b, r, and a for rectangular and circular robot footprints (see Examples 50 and 51). The left subfigure shows a rectangular footprint, with length L and width W. The right subfigure shows a circular robot footprint with diameter $2R$. The maximum penetration distance $\bar{b}$ is omitted for clarity.

4.2.2.4 Constructing the discretized Prediction

Now I present an algorithm to take a buffered obstacle and discretize its boundary, producing the discretized obstacle X_p. Theorem 52 proves that, if the robot cannot collide with any point in X_p, then it also cannot collide with the obstacle. Finally, sources of conservatism in the discretization approach are discussed.

To get the buffered obstacle, let X_{obs} consist of polygons (as in Assumption 43). Suppose X_0 is the robot's footprint at time 0 (as in Definition 12), which is compact and convex with nonzero volume (as in Assumption 42). Suppose that $\bar{r}$ as in Definition 49 and $\bar{b}$ is given as discussed in §4.2.2.2. Select $b \in (0, \bar{b})$, then determine r and a. Buffer the obstacle to produce P_b as in (4.15). Now, ∂P_b is discretized.

The first two functions extract the lines and arcs from the boundary of the buffered obstacle. Then, by Lemma 47, $\partial P_b = L \cup A$ where L is a finite set of closed line segments and A is a finite set of closed arcs. Let n_L be the number of line segments and n_A be the number of arcs. For $i = 1, \ldots, n_L$, let $L_i \subset L$ denote the i^{th} segment, and similarly $A_i \subset A$ for the i^{th} arc. Then the function `extractLines` takes in the buffered obstacle P_b and returns the set L of all line segments on ∂P_b. Similarly, the function `extractArcs` takes in P_b and returns the set A of all circular arcs on ∂P_b.

A third function, `sample`: $\mathcal{P}(\mathbb{R}^2) \times \mathbb{R} \to \mathcal{P}(\mathbb{R}^2)$, is defined to discretize the line segments and arcs. Suppose $S \subset \mathbb{R}^2$ is a connected curve with exactly two endpoints and no self-intersections. Let $s > 0$ be a distance. Then $P = \texttt{sample}(S, s)$ is a set containing the endpoints of S. Furthermore, if the total arclength along S is greater than s, then P also contains a finite number of

points spaced along S such that, for any point in P, there exists at least one other point that is no farther away than the arclength s along S. Note that the line segments in L and the arcs in A can be parameterized; then the `sample` function can be implemented using interpolation of a parameterized curve.

Algorithm 2 Construct Discretized Obstacle (`discretizeObs`)

Require: $P_b \subset X$, $r \in \mathbb{R}_{\geq 0}$, $a \in \mathbb{R}_{\geq 0}$
$L \leftarrow$ `extractLines` (P_b), $A \leftarrow$ `extractArcs` (P_b)
$X_p \leftarrow \emptyset$
For each: $i \in \{1, \ldots, n_L\}$
    $X_p \leftarrow X_p \cup$ `sample`(L_i, r)
end
For each: $j \in \{1, \ldots, n_A\}$
    $X_p \leftarrow X_p \cup$ `sample`(A_j, a)
end
Return X_p

Suppose that X_p is constructed from a buffered prediction P_b using Algorithm 2. Then X_p contains the endpoints of each line segment or arc of ∂P_b, since it is constructed using `sample`. In addition, for each line segment of ∂P_b, X_p contains additional points spaced along the line segment such that each point is within the distance r (in the 2-norm) from at least one other point. Similarly, for each arc of ∂P_b, X_p contains points spaced along the arc such that each point is within the arclength a of at least one other point; this implies that distance between any pair of adjacent points along each arc is no more than a. Finally, note that $|X_p|$ is finite, because there are a finite number of polygons in P (see Assumption 43), each polygon has a finite number of edges, and $r, a > 0$.

4.2.2.5 Proving Safety

Now, the notion that X_p represents the obstacles P is formalized. Recall that the purpose of constructing X_p is to write a finite list of ineqaulity constraints for the trajectory optimization program. To ensure safety, the set $\pi_K(X_p)$ must contain all possible unsafe trajectory parameters $\pi_K(P)$, which is the complement of the set K_{safe}, leading to the following theorem:

Theorem 52. *Let X_0 be the robot's footprint at time 0 as in Definition 12, with width $\bar{r}$ as in Definition 49. Let $P \subset (X \setminus X_0)$ be a set of predictions as in Definition 31. Suppose that the maximum penetration depth $\bar{b}$ is found for X_0 as in Lemma 46. Pick $b \in (0, \bar{b})$, and find the point spacing r and the arc point spacing a. Construct the discretized obstacle X_p in Algorithm 2. Then, the set of all unsafe trajectory parameters corresponding to P is a subset of the trajectory parameters corresponding to X_p, i.e. $\pi_K(X_p) \supseteq \pi_K(P)$.*

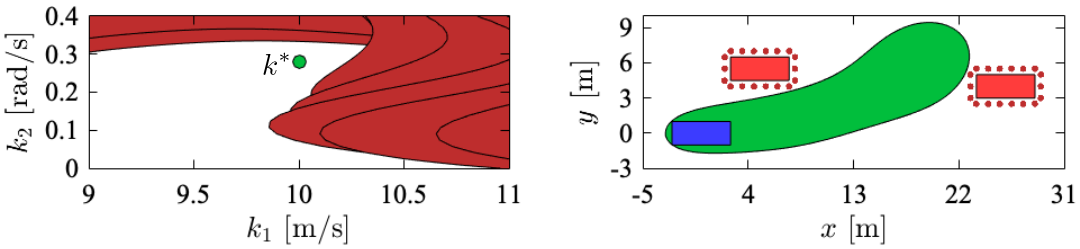


Figure 4.6: The right plot shows the 1 superlevel set of w, $\pi_X(k^*)$, in green, and the obstacle discretization, X_p, for two polygon obstacles as red points. The left plot shows the projection of X_p into the parameter space $\pi_K(X_p)$ in red and the parameter k^* as a green point. The reachable set computation is described in §5.2.2. The buffer distances are $b = 0.05$ m, $r = 0.1$ m, and $a = 0.07$, computed with Example 50.

Proof. Provided in Appendix B □

Theorem 52 provides the main result of this section: $\pi_K(X_p)^C \subseteq K_{\text{safe}}$. In other words, one can use X_p to inner approximate K_{safe} by evaluating the w polynomials as in (4.12). Figure 4.6 shows the resulting discretization X_P and sets of unsafe parameters $\pi_K(X_p)$ for the reachable set of the autonomous vehicle model described in §5.2.

4.3 Dynamic Obstacles

This section focuses on extending the discrete obstacle representation presented in §4.2.2 to dynamic obstacles. Each of these methods will develop constraints at a set of discrete points $X_p \subset X$ for their collision checks, i.e. Φ will resemble (4.12). Two methodologies are covered. §4.3.1 performs a collision check at discrete time points and guarantees collision avoidance between time steps by formulating a buffer [VKL+19]. Unfortunately, the buffer must be applied uniformly; which can greatly reduces the free space a robot can operate in. §4.3.2 performs a collision check over time intervals, which eliminates the need to buffer for time discreteization [VKL+19]. The time interval approach is preferable since conservatism is introduced along the trajectories of the robot and prediction as oppose to being applied uniformly.

4.3.1 Collision Check at Discrete Time Points

This section covers a method to perform collision checks at discrete time points. Assume that predictions are given as in Definition 31 and Assumption 43. A time discretization is selected to perform a collision check at, meaning the trajectory optimization will ensure collision avoidance a

finite set of timepoints

$$T_{\text{disc}} = \{j\tau_{\text{disc}}\}_{j=0}^{n_{\text{pred}}}, \tag{4.17}$$

where jn_{pred} is equal to the time horizon T. Enforcing the collision check at timepoints in T_{disc}, does not account for movement by the ego vehicle and robot between timesteps. A buffer b_{disc} must be added to each prediction, where buffer is defined as in Definition 45, to account for all possible actions that can happen between timesteps

Lemma 53. *Pick $b \in (0, W/2)$ and a time discretization τ_{disc}. Select a* temporal buffer*, b_{disc}, satisfying*

$$b_{\text{disc}} \geq \tau_{\text{disc}}(v_{\text{max}} + v_{\text{obs,max}})/2, \tag{4.18}$$

where v_{max} and $v_{\text{obs,max}}$ are the maximum speed of robot and obstacle. Suppose $P_{b_{\text{disc}}}$ is a prediction. Suppose the current time is $t_1 \in [0, T - \tau_{\text{disc}}]$, let $t_2 \in t_1 + (0, \tau_{\text{disc}})$. If the robot does not intersect the prediction $P_{b_{\text{disc}}}(t_1)$ at t_1 or $P_{b_{\text{disc}}}(t_1 + \tau_{\text{disc}})$ at $t_1 + \tau_{\text{disc}}$. Then it does not intersect the prediction $P_{b_{\text{disc}}}(t_2)$ at t_2.

Proof. Note the existence of v_{max} follows from Assumptions 2 and $v_{\text{obs,max}}$ is given by 32. By definition 45, and the fact that it does not intersect the prediction, the closest that the robot can be to any obstacle at time t_1 is greater than b_{disc}. Similarly, the closest it can be at time $t_1 + \tau_{\text{disc}}$ is greater than b_{disc}. So, for the robot to collide with any obstacle over $[t_1, t_1 + \tau_{\text{disc}}]$, the robot must travel strictly more than $2b_{\text{disc}}$ relative to the obstacle. By construction of (4.18), this is not possible unless the robot or the obstacle exceeds its maximum speed. □

A similar discretization method from Algorithm 2 is applied to the prediction at each timestep and collision check against a time varying forward-reachable set. However, unlike the static obstacle case, the assumption that the robot enters predictions from the outside is no longer valid. To construct a set of discrete points for the interior, notice that by Assumption 43, at time t, the set $P_b(t) \subset X$ is the union of a finite set of closed polygons. Define a function $\texttt{cover}{:}\mathcal{P}(X) \times \mathbb{R}_{\geq 0} \to X$ as follows. Given $P_b(t)$, a point spacing r, and the width of the robot footprint, W, let $\texttt{cover}(P_b(t), W)$ return a (finite) set $A \subset X$, such that, points in A are contained within P_b, and if $B_{W/2}(a)$ is a 2-norm ball of radius $W/2$ centered at a, then there exists $a' \in A \setminus \{a\}$ for which $a' \in B_{W/2}(a)$. Furthermore, the union of all such balls covers $P_b(t)$ (i.e., $\bigcup_{a \in A} B_{W/2}(a) \supseteq P_b(t)$).

In short, Algorithm 3 returns points with a maximum spacing r along the boundary of $P_b(t)$, and points in the interior with a maximum spacing of $W/2$, where W is the width of the robot. Figure 4.7 shows a cartoon of the FRS being intersected with such predictions.

Algorithm 3 Construct Discretized Obstacle with Interior Points (`discretizeObsWithInterior`)

Require: $P_b \subset X, r \in \mathbb{R}_{\geq 0}, a \in \mathbb{R}_{\geq 0}, W \in \mathbb{R}_{\geq 0}$
$L \leftarrow \texttt{extractLines}(P_b)$, $A \leftarrow \texttt{extractArcs}(P_b)$
$X_p \leftarrow \emptyset$
For each: $i \in \{1, \ldots, n_L\}$
    $X_p \leftarrow X_p \cup \texttt{sample}(L_i, r)$
end
For each: $j \in \{1, \ldots, n_A\}$
    $X_p \leftarrow X_p \cup \texttt{sample}(A_j, a)$
end
$X_p \leftarrow X_p \cup \texttt{cover}(P_b, W)$
Return X_p

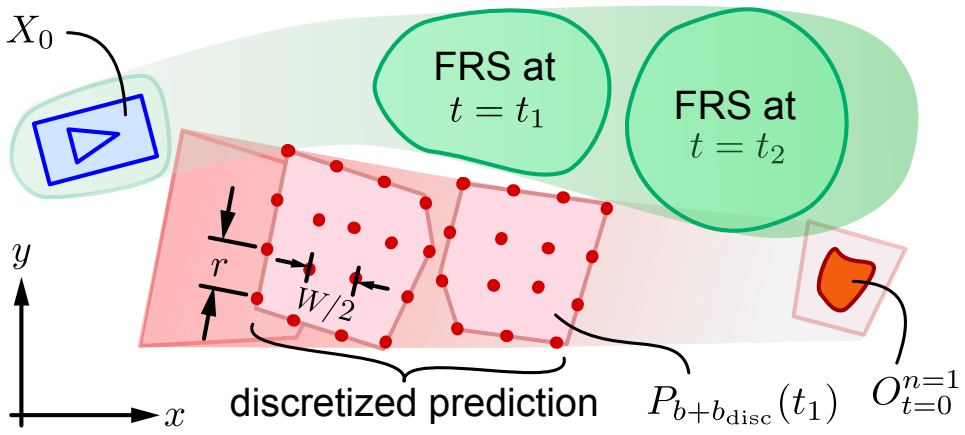


Figure 4.7: Discretization of a prediction $P_{b+b_{\text{disc}}}$ as in Theorem 54. The robot plans a not-at-fault trajectory for any $t \in T$ given the prediction (right to left). The FRS is shown left to right by the 1 superlevel set of w from (D_T). Temporal discretization is shown at two times, t_1 and t_2; at each time, the prediction is spatially discretized with points along the boundary space r apart and points in the interior space $W/2$ apart, as in Algorithm 3, where W is the ego robot's width.

Theorem 54. *Let (v, w, q) be a feasible solution to (D_T) from §3.3.1, τ_{disc} be a time discretization as specified in* (4.17)*, b_{disc} be as in 4.18, and the quantities b, r, a slected as in §4.2.2.2. Let W be the width of the vehicle. Then for $k \in K$:*

$$w(j\tau_{\text{disc}}, p_j, k) - 1 \leq 0,\ \forall p_j \in X_p(t_j),\ j \in \{0, \ldots, n_{\text{pred}}\} \implies k \in K_{\text{safe}} \tag{4.19}$$

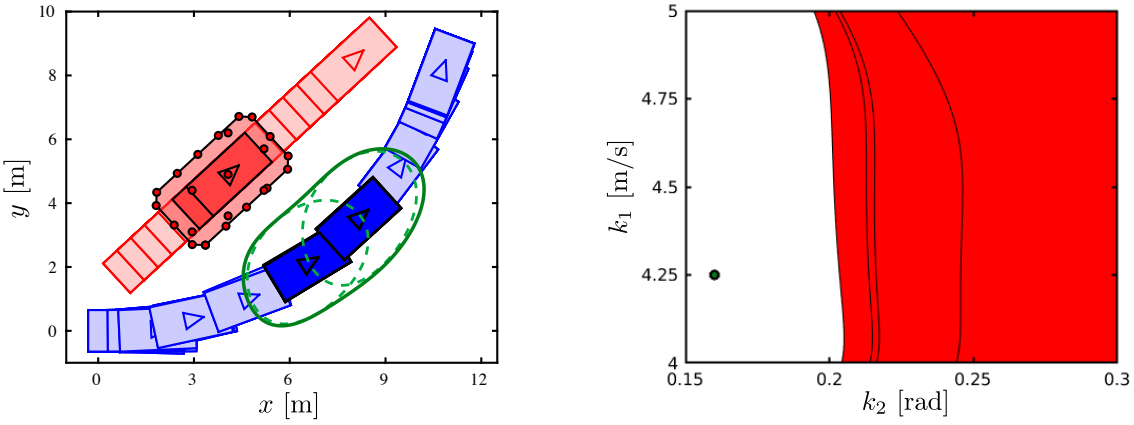
where $X_p(t_j)$ is generated by applying Algorithm 3 to buffered predictions at the j^{th} timestep. The predictions, $P_{b+b_{\text{disc}}}(t_j)$, are constructed as in Definition 45.

Proof. First note that (4.19) guarantees that the robot will not be in collision with the predictions at any $t \in T_{\text{disc}}$. This follows directly from first applying Theorem 52, which indicates that a robot lying outside of the obstacle cannot intersect both the prediction and obstacle. The points in the interior ensure that it is not possible for the robot to lie within the prediction without $w(j\tau_{\text{disc}}, p_j, k) \geq 1$ for at least one $p_j \in X_p(t_j)$. I prove this by contradiction. Let $k \in K$. Suppose for contradiction that no $p_j \in \pi_X(k)$. Then there exists $x \in \pi_K(X)$ for which $\|x - p_j\|_2 > W/2$, since the 1 superlevel set of w contains the vehicle footprint at t_j; but then $P_{b+b_{\text{disc}}}(t_j)$ is not covered by balls of radius $W/2$ as defined in `cover`. Lemma 53 provides that there will be no collision at any $t \in [0, T] \setminus T_{\text{disc}}$. □

4.3.2 Collision Check with Time Intervals

Although the discrete time collision check discussed in §4.3.1 is provably safe, it can be conservative since the buffer b_{disc} has to be applied uniformly around each prediction. Consider an autonomous driving as an example, where the maximum speeds of the robot and obstacles, v_{max} and $v_{\text{obs,max}}$, are high. The high speeds increase the buffer b_{disc} as per (4.18); which is applied uniformly to each prediction, meaning it decreases free space in both the longitudinal (road aligned) and lateral directions. With vehicles the longitudinal velocities are generally large compared to the lateral velocities, so developing a collision check that takes this directionality into account is motivated; since vehicles operate in laterally narrow lanes. In this section collision checks are performed over *time intervals* instead of at discrete time points. In this case, conservatism is introduced along the trajectories of the robot and obstacles, which §5.3 shows is favorable to applying uniform buffers for driving applications.

For out collision check, first specify a collection of closed *time intervals* $\mathbf{t}_1, \ldots, \mathbf{t}_{n_I}$, $n_I \in \mathbb{N}$, that cover T, meaning $\cup_{j=1}^{n_I} \mathbf{t}_j = T$. Each FRS interval can be written as $\mathbf{t}_j = t \in [t_j, t_{j+1}]$; note that $\mathbf{t}_1 = [0, t_1]$ and $\mathbf{t}_{n_I} = [t_{n_I-1}, T]$. Reachable points $(x, k) \in X \times K$, independent of time, are computed for each FRS interval, and predictions of obstacle motion during each interval are treated as static. Since the FRS intervals cover T, there is no buffer to compensate for points that



(a) (b)

Figure 4.8: Subfigure (a) shows trajectories of an obstacle (red rectangles) and the ego vehicle (blue rectangles) for a time horizon of $T = 7.5$ s. The ego vehicle's trajectory is a solution to (3.2). The 0-sublevel set of v (green dashed) and 1-superlevel set of w_j (green solid) are shown for the time interval $\mathbf{t}_j = [2.5, 3.0]$ s. The obstacle prediction (light red) and discretization (dark red) are shown for the same interval. The white area of Subfigure (b) is the projection of the discrete points $X_p(\mathbf{t}_j)$, into the (k_1, k_2) subspace. The trajectory parameter used by the ego vehicle is shown as a dot.

are reachable between discrete times. The FRS in each FRS interval, $\mathcal{X}_{\text{FRS},j}$ (3.46), is described by a polynomial $w_j : X \times K \to \mathbb{R}$ for each $\mathbf{t}_j$ such that, if $t \in \mathbf{t}_j$ and $\mathcal{X}_{\text{FRS},j}$, then $w_j(x, k) \geq 1$. See program (D_I) in §3.5 and §3.3.2 for an explanation of how to compute such a polynomial.

Each FRS is collision checked against a prediction containing the obstacles' motion over a matching time interval. This requires a modified definition of predictions

Definition 55. *A* conservative prediction over a time interval *is a map* $P : [t_1, t_2] \to \mathcal{P}(X)$*, with* $0 \leq t_1 \leq t_2$ *that contains all obstacles within* D_{sense} *(Assumption 30) at any time* $t \in [t_1, t_2]$*; i.e.,* $P([t_1, t_2]) \supseteq \bigcup_{i \in \{1,\dots n_{\text{obs}}\}, t \in [t_1, t_2]} O_t^i$*. For notational convenience, write* P *evaluated for an interval* $\mathbf{t}_j$ *as* $P(\mathbf{t}_j)$

As in Assumption 43, assume that each prediction is a union of a finite number of closed polygons. Then apply the same discretization as described in Algorithm 3 to each time interval $\mathbf{t}_j$ to ensure that the FRS of the robot does not intersect the prediction of the obstacle. The proof of collision avoidance for each time interval is the same as Theorem 54, without the buffer for time discretization. Enforcing this constraint on all $\mathbf{t}_j$, ensures the robot is collision free for all $t \in [0, T]$. A depiction of the collision check for one time interval $\mathbf{t}_j$, along with unsafe parameters given by $\pi_K(X_p(\mathbf{t}_j))$ is shown in Figure 4.8.

CHAPTER 5

Reachability-based Trajectory Design: Demonstrations and Comparisons

This chapter presents demonstrations of RTD and comparisons of RTD against state-of-the-art motion planning algorithms. §5.1 compares RTD against rapidly exploring random tree (RRT) and nonlinear model predictive control (NMPC) algorithms in simulations with static obstacles using the Segway and Rover in Figure 3.2. Hardware demonstrations with each platform are also provided. §5.2 provides an implementation of RTD on a Carsim model of a Ford Fusion, navigating a test track with static obstacles. §5.3 compares RTD against state lattice and MPC planners in simulations with dynamic obstacles. The Segway, and a small electric vehicle are used as platforms for the simulation and hardware demonstrations.

5.1 Static Obstacles

This section details the application of RTD to two robots: the Segway (Figure 3.2a), and the Rover (Figure 3.2b). Both robots use Algorithm 1 (§4.1) for online safe trajectory planning, demonstrated in simulation (§5.1.4) and on hardware (§5.1.5).

5.1.1 Segway

This section describes the Segway platform and its environment used in the simulation and hardware sections.

5.1.1.1 Platform

The Segway is a differential-drive robot. RTD is applied to the Segway to show that the proposed method can provide collision-free trajectory planning in unstructured, random environments. The

Segway has been used as a running example through this paper. Example 4 presents its high-fidelity model (3.5).

Next, the Segway's model parameters are described. The robot has a circular footprint with a 0.38 m radius. It is limited to a maximum yaw rate $|\dot{\theta}| \leq 1$ rad/s and a maximum speed of $v \leq 1.5$ m/s in simulation and $v \leq 1.25$ m/s on the hardware. The acceleration bounds are $[\underline{\gamma}, \overline{\gamma}] = [-5.9, +5.9]$ rad/s^2, and $[\underline{\alpha}, \overline{\alpha}] = [-3.75, 3.75]$ m/s^2. Given a current yaw rate, $\dot{\theta}$, the commanded yaw rate, $\dot{\theta}_{\text{des}}$, $|\dot{\theta} - \dot{\theta}_{\text{des}}| \leq 1$ rad/s is enforced in simulation, and $|\dot{\theta} - \dot{\theta}_{\text{des}}| \leq 0.5$ rad/s on the hardware. Motion capture data is used to find the parameters $\beta_\gamma = 2.95$, $\beta_\alpha = 3.00$. State estimation error, as described in §3.2.3 has to be considered for the hardware platform.

5.1.1.2 FRS Computation

For the FRS computation, Example 5 presents the trajectory-producing model, Figure 3.3 shows its tracking error function, and Example 51 presents the geometric quantities needed to represent obstacles for the Segway.

The FRS is computed by solving (D^l) in §3.3.2, with $l = 5$. In practice, the tracking error is proportional to the initial speed, so computing multiple FRS's reduces conservatism. At runtime, the appropriate FRS is chosen based on the Segway's estimated initial speed at the beginning of the current planning iteration. For the simulations, one FRS for each of the following initial speed ranges: 0–0.5 m/s, 0.5–1.0 m/s, and 1.0–1.5 m/s is computed. For the hardware, FRS's for initial speed ranges of 0.0–0.5 m/s and 0.5–1.25 m/s are computed. In simulation, the FRS is computed over a time horizon of $T = 0.6$ s for the 0.0–0.5 m/s FRS, and $T = 0.8$ s for the other two FRS's. For the hardware, the Segway's FRS is computed over a time horizon $T = 1$ s. For all of the Segway FRS's, $\tau_{\text{plan}} = 0.5$ is used.

The following geometric quantities (as introduced in Section 4.2.2.2) are used to represent obstacles for the Segway. The width of the Segway is $\bar{r} = 0.76$ m (Definition 49) and the maximum penetration distance is $\bar{b} = 0.38$ m (Lemma 46). In the simulations, a buffer size of $b = 0.001$ m is empirically. This choice of b results in a point spacing $r = 0.055$ m and arc point spacing $a = 0.002$ m as per Definition 48 and Example 51. On the hardware, a buffer size of $b = 0.05$ m is used, so $r = 0.37$ m and $a = 0.10$ m. Recall from above that $\tau_{\text{plan}} = 0.5$. The choice of buffer b was the smallest buffer that allowed the runtime trajectory optimization to solve consistently within τ_{plan} (recall that the number of constraints for trajectory optimization increases as b decreases).

5.1.1.3 Environment

The simulated environment for the Segway is a 9×5 m^2 room, with the longer dimension oriented east-west. The room is filled with 6 to 15 randomly-distributed box-shaped obstacles with a side

length of 0.3 m. A random start location is chosen on the west side of the room and a random goal is chosen on the east side. The simulated environment is similar to the hardware demo depicted in Figure 3.2a. A trial is considered successful if the Segway reaches the goal without crashing (i.e., touching any obstacles). Since obstacles are distributed randomly, it may be impossible to reach the goal in some trials; hence the number of crashes and number of goals reached are counted separately.

For the Segway's high-level planner, Dijkstra's algorithm is used to find the shortest path on a graph representing a grid in the robot's xy-subspace X; this provides a coarse path and intermediate waypoints between the Segway and the global goal. At each planning iteration, the cost function given to Algorithm 1 Line 7 attempts to minimize the distance to the current waypoint. Additionally, for all planners, if the Segway has braked to a stop without crashing, one planning iteration is spent rotating in place towards the current waypoint before replanning.

5.1.2 Rover

This section describes the Rover platform in Figure 3.2b and its environment used in the simulation and hardware sections.

5.1.2.1 Platform

The Rover is a front wheel steering, all-wheel drive platform, and demonstrates the utility of RTD in passenger robot applications. The trajectory producing model is presented in Example 20. The system decomposition technique discussed in Section 3.4 is used to compute the FRS's.

The Rover has a rectangular footprint of length 0.5 m and width 0.29 m centered at the center of mass. The distance from the rear axle to the center of mass, l_r, is 0.0765 m. The Rover's high-fidelity model has a state vector $z_{\text{hi}} = [x, y, \theta, v_x, \delta]^\top$, where v_x is longitudinal speed and δ is the angle of the front (steering) wheels relative to the Rover's longitudinal direction of travel. The dynamics f_{hi} as in (3.1) are:

$$\frac{d}{dt}\begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \\ v_x(t) \\ \delta(t) \end{bmatrix} = \begin{bmatrix} v_x(t)\cos(\theta(t)) - \dot{\theta}(t)(c_1 + c_2 v_x(t)^2)\sin(\theta(t)) \\ v_x(t)\sin(\theta(t)) + \dot{\theta}(t)(c_1 + c_2 v_x(t)^2)\cos(\theta(t)) \\ \frac{v_x(t)}{c_3 + c_4 v_x(t)^2}\tan(\delta(t)) \\ c_5 + c_6(v_x(t) - u_1(t)) + c_7(v_x(t) - u_1(t))^2 \\ c_9(u_2(t) - \delta(t)) \end{bmatrix}. \tag{5.1}$$

This model utilizes steady-state assumptions for the lateral dynamics, but the constants c_2 and c_4 account for wheel slip [SHB14, §10.1.2]. Motion capture data was used to fit the constants, c. The

steering wheel angle input, u_1, is bounded by $|\delta| \leq 0.5$ rad, and the speed input, u_2, is manually limited to 0 to 2 m/s.

5.1.2.2 FRS Computation

For the rover, the tracking controller is a proportional controller; hence the dynamics (5.1) are Lipschitz continuous in t, z_{hi}, and u as required by Assumption 2. Example 20 presents the Rover's trajectory-producing model (3.30). Recall that the trajectory-tracking model (3.13), is the trajectory-producing model plus the tracking error functions (as in Assumption 7). The trajectory-tracking model dynamics $\dot{z}_i : [0, T] \times K \to \mathbb{R}^2$ (as in (3.13)) for each SCS are given by:

$$\dot{z_1}(t) = \begin{bmatrix} k_1(1 - \frac{\theta(t)^2}{2}) - l_r\dot{\theta}(t,k) \cdot (\theta(t) - \frac{\theta(t)^3}{6}) \\ \dot{\theta}(t,k) \end{bmatrix} + \begin{bmatrix} g_1(t,k) \\ 0 \end{bmatrix} d_1(t) \tag{5.2}$$

$$\dot{z_2}(t) = \begin{bmatrix} k_1(\theta(t) - \frac{\theta(t)^3}{6}) + l_r\dot{\theta} \cdot (1 - \frac{\theta(t)^2}{2}) \\ \dot{\theta}(t,k) \end{bmatrix} + \begin{bmatrix} g_2(t,k) \\ 0 \end{bmatrix} d_2(t) \tag{5.3}$$

where: $g_x, g_y \in \mathbb{R}_3[t,k]$ are degree 3 polynomials that satisfy (3.16); the yaw rate $\dot{\theta}(t,k) = \frac{-k_3}{2}t + k_2(1-t)$ is given by (3.31) with $T_h = 2$ s; and $d_1, d_2 : [0,T] \to [-1,1]$ are scalar-valued functions. The tracking error functions in (5.2) and (5.3) are fit to trajectory data as with the Segway.

For the Rover, $(D_T^{(i),4})$ is solved for the subsystems 1 and 2 in (3.39) and (3.40). Then, (R^5) reconstructs the full system FRS. As with the Segway, the tracking error for the Rover is reduced by computing multiple FRS's, each corresponding to a different range of initial conditions. 42 FRS's are computed for the Rover in total. Each FRS has one of three ranges of initial speeds: 0.0–0.75 m/s, 0.75–1.5 m/s, and 1.5–2.0 m/s; one of seven ranges of initial wheel angles evenly spaced between -0.5 and 0.5 rad; and either positive or negative headings.

The Rover selects an FRS at runtime based on its initial velocity, wheel angle, and heading at each planning iteration. The time horizons are $T = 1.25$ s for the slowest FRS's and $T = 1.5$ s for the faster FRS's. All FRS's use $\tau_{\text{plan}} = 0.5$ s in simulation. On hardware, $\tau_{\text{plan}} = 0.375$ s and one FRS with velocities between 1 and 1.5 m/s is used due to the limited size of the physical testing area available. The range of trajectory parameters for each FRS is determined as follows: The final headings, k_3, are between 0 and 0.5 (resp. -0.5) rad for FRS's with negative (resp. positive) initial headings. The initial yawrates, k_2, are between $\max(-1, -1+2k_3)$ and $\min(1, 1+2k_3)$ rad/s. The desired velocities, k_1, are set so the change between initial and commanded velocity is less than 1 m/s, and a minimum of 0.5 m/s for the slowest FRS.

A buffer $b = 0.01$ m is used for the Rover, resulting in the point spacing $r = 0.02$ m and arc point spacing $a = 0.014$ m as per Example 50.

5.1.2.3 Environment

The simulated environment for the Rover is a larger version of the mock road depicted in Figure 3.2b, which mimics a highway environment. The simulated road lies along the x-direction (oriented east-west) and is centered at $y = 0$. It is 2.0 m wide (including the shoulder), with two 0.6 m wide lanes centered at $y = 0.3$ m and $y = -0.3$ m. The Rover plans trajectories with speeds up to 2 m/s. In each trial, three randomly sized box-shaped obstacles of lengths 0.4–0.6 m and widths 0.2–0.3 m are placed in alternating lanes. This obstacle arrangement is used to force the Rover to attempt two lane changes per trial; note that the RTD, RRT, and NMPC trajectory planners are all general implementations (§5.1.3.1- 5.1.3.3), not specialized to this particular obstacle arrangement. The obstacles have a random heading of $\pm$ 2 degrees relative to the road, and their centers are allowed to vary by $\pm$ 0.1 m from lane center in the y-dimension. The spacing between the obstacles in the x-direction is given by a normal distribution with a mean of 4 m and standard deviation of 0.6 m. The Rover begins each trial centered in a random lane, with a velocity of 0 m/s. A trial is considered successful if the Rover crosses a line positioned 30 m after the third obstacle without intersecting any obstacle or road boundary (i.e. crashing).

For high-level path planning, a desired waypoint is placed a set distance ahead of the robot and centered in the current lane. If the waypoint is inside or behind an obstacle relative to the Rover, the waypoint is switched to the other lane. It was found empirically that placing the waypoint 4 m ahead of the Rover at each planning iteration causes it to switch lanes soon enough the Rover is typically capable of performing a lane change; this 4 m "lookahead distance" was used for all three planners.

5.1.3 Trajectory Planner Implementations

This section describes the implementations of RTD, a Rapidly-exploring Random Tree (RRT) planner based on [KTF+09, PKA16, PLM06], and a GPOPS-II Nonlinear Model-Predictive Control (NMPC) planner [PR14]. This section is organized as follows. §5.1.3.1 describes the implementation of RTD. §5.1.3.2 describes the implementation of the RRT. §5.1.3.3 describes the implementation of GPOPS-II.

5.1.3.1 RTD Implementation Details

This section discusses particular implementation details used for RTD in the simulations. The following cost functions are used in each planning iteration. For the Segway, the cost function is the robot's Euclidean distance at time T to the waypoint generated by the high-level planner. For the Rover, the cost function is the Euclidean distance at time T_h from the planned trajectory's endpoint to the waypoint, weighting error in x vs. error in y at a ratio of 1:2. The final heading

parameter, k_3 in (3.31), is set to be the negative of the Rover's initial heading (saturated at $\pm$ 0.5 rad), so the Rover only optimizes over k_1 and k_2 in each iteration. In other words, the Rover only optimizes over trajectories that will align the robot with the road.

Both the Segway and Rover use the static obstacle discretization method in §4.2.2 to generate constraints for collision avoidance in Algorithm 1 Line 6 (the quantities of interest are described in §5.1.1.2 and §5.1.2.2). Both the Segway and the Rover use MATLAB's `fmincon` generic nonlinear solver to implement the online trajectory optimization (Algorithm 1 Line 7). For both robots, an optimality tolerance of 10^{-3} is used. Since `fmincon` is a generic gradient-based nonlinear solver, it requires an initial guess each time it is called (i.e., in each planning iteration). For the Segway, the initial guess of $k \in K$ corresponds to zero yaw rate and maximum speed. For the Rover, the initial guess is either the trajectory parameters from the previous planning iteration, or parameters corresponding to driving straight if the previous iteration converged to an infeasible result.

The following design choice to speeds up `fmincon`. Recall from Section 4.2.2 that obstacles are represented as sets of discrete points. Each discrete obstacle point becomes a nonlinear constraint for `fmincon`. Since `fmincon`'s solve time increases with the number of constraints, the number of constraints in each planning iteration can be reduced by discarding points in X_p that lie outside of the FRS for any trajectory parameter $k \in K$. Note that, since no such points are reachable (because they lie outside of the FRS), this does not impact RTD's safety guarantees.

5.1.3.2 RRT Implementation Details

The Segway and Rover use similar RRT implementations, based on several papers [KTF+09, PKA16, PLM06], which describe a variety of heuristics for growing a tree of a robot's trajectories with nodes in the high-fidelity state space. Both RRT implementations use the entire duration τ_{plan} to plan a trajectory at each planning iteration.

To account for the robot's footprint, obstacles are buffered by Minkowski sum with a polygonal outer approximation of a closed disk, with radius given by the desired buffer distance (see Experiment 1 in Section 5.1.4.1 for how to empirically select the buffer). This produces a representation of each buffered obstacle as a collection of half-planes.

For the Segway, the RRT planner begins by checking if the previously-planned trajectory is still feasible [KTF+09], meaning that none of its nodes lie inside any buffered obstacles. If the past trajectory is feasible, the tree is initialized with the previous plan's nodes; if the past trajectory is infeasible, the tree is initialized from the robot's initial state. For the Rover, which operates in a simpler environment, a new tree was initialized for every planning iteration. New nodes of the tree are created by first choosing a random existing node, with the choice biased towards more recently-generated nodes. From the randomly-chosen node, the high-fidelity robot model is

forward-integrated under a random desired yaw rate (or wheel angle) and desired speed [KTF+09, PLM06]. Forward-integration of the high-fidelity model dynamics returns points in the robot's xy-subspace X at a set of discrete time points. A new node is discarded if any of these points lie inside any buffered obstacle, outside of the robot's environment (the room for the Segway and the road for the Rover), or outside the robot's sensor horizon. In addition, for the Segway, recall (from §5.1.1.3) that Dijkstra's algorithm is used for generating a high-level plan; nodes farther than 1.5 m from the high-level plan are discarded [PKA16]. For both the Segway and the Rover, the RRT attempts to plan a braking maneuver at each planning iteration.

Forward integration of the robot's high-fidelity model is required for dynamic feasibility of the RRT trajectory plans, given the complexity of the high-fidelity models of the Segway and Rover [ES14]. The *edge time*, or total duration of each forward-integration, along with the time discretization, are heuristic choices that affect the computation time and complexity of paths that the RRT can generate; these numbers were selected empirically for each system. This implementation makes use of MATLAB's symbolic and function generation toolboxes. For the Segway, an integration function is generated that takes in an initial condition and returns a trajectory of the robot's high-fidelity model, forward-integrated with an RK4 method, for a predetermined edge time and step size. The timing of calls to this function was tested by forward integrating each robot's high-fidelity model from random initial conditions and compared to that using the ODEINT C++ library [AM11]. For the Rover, forward Euler integration is used, and it was able to plan safely (see §5.1.4.1).

Recall that, for both the Segway and Rover, a high-level planner generates intermediate waypoints. When growing the RRT, samples are biased to turn towards waypoints as described by [KTF+09]. For the Segway, the RRT attempts to find a plan that minimizes distance to the waypoint. For the Rover, minimizing distance to the waypoint resulted in the RRT generating long paths with large changes in yaw rate because path smoothness was not included in the cost. To combat this, the RRT's cost at each node was set as the cumulative distance from the root node, plus a penalty for lying close to obstacles [KTF+09], reduced the number of crashes. Once the RRT has grown for the duration τ_{plan}, the node with the lowest cost is chosen to produce the trajectory plan. The Rover's RRT has an additional heuristic to encourage smoothness: when the waypoint is in the same lane as the rover, the standard deviation of sampled wheel angles is reduced.

5.1.3.3 NMPC Implementation Details

The Segway and Rover both use GPOPS-II for the nonlinear model predictive control planner [PR14]. GPOPS-II is an algorithm that approximates the trajectory planning problem as a polynomial optimization program. This software uses internal heuristics to choose a finite number of collocation points, then evaluates the polynomial approximation of the robot's high-fidelity model

and obstacle avoidance constraints at each of these points. The accuracy of the solution and the run time of the algorithm is dependent on the tolerance of the polynomial approximation. The cost function used at each planning iteration is to minimize distance between the last collocation point and the waypoint generated by the high-level planner.

The following constraints are enforced each planning iteration. Obstacles are represented as constraints on the x and y coordinates of the robot's center of mass at each collocation point. Each obstacle is buffered using a Minkowski sum with a polygonal outer-approximation of a closed disk with radius given by a user-selected buffer distance (see Section 5.1.4.1). This representation means that, to check for collision of a trajectory with an obstacle, a finite number of half-plane checks are performed per obstacle per collocation point. The maximum speed and yaw rate of each robot are also enforced constraints.

A fail-safe for NMPC is encoded in the following manner: if no feasible trajectory can be found within τ_{plan}, the robot continues executing the last feasible trajectory that NMPC found. For the Segway, an additional constraint requires the end of any planned trajectory must have zero speed and yaw rate, to force NMPC to plan a braking maneuver. For the Rover, the minimum time horizon is set to 1.5 s, (the braking time from 2 m/s); although potentially less robust than the Segway's constraint, it was sufficient for the environment the Rover is tested in.

The decision variables for NMPC are the robot's state and control input at each collocation point. For the Segway, the NMPC planner chooses a desired yaw rate and velocity as the control input at each collocation point, and plans with the robot's high-fidelity model (3.5) from Example 4. For the Rover, the NMPC planner chooses a desired wheel angle and velocity as the control input at each collocation point, and plans with the robot's high-fidelity model (5.1).

GPOPS-II is initialized at each planning iteration as follows. The planner is given a coarse trajectory guess at the first planning iteration, and each subsequent iteration is seeded with last feasible trajectory. The GPOPS-II parameters used are: 4–10 collocation points per phase and a mesh tolerance of 10^{-6}.

5.1.4 Simulation Results

This section compares RTD, against the RRT and NMPC planners described in §5.1.3. The contribution of this section is the comparison of RTD to RRT and NMPC, and the demonstration of safety of RTD over thousands of simulations. Code used in the simulations is available at `https://github.com/skvaskov/RTD`.

The timing parameters, environments, and high-level planners used for the simulations are now discussed. Recall the planning hierarchy introduced in §1.1.1. RTD is a trajectory planner, in the middle level of the hierarchy; therefore, RTD's role is to plan trajectories that attempt to

achieve a coarse path plan generated by a high-level planner. In this work, the high-level planner generates intermediate waypoints, or desired locations, between the robot and the global goal. These waypoints define the cost function for trajectory optimization in each planning iteration.

As in Algorithm 1, the robot is limited by a physical sensor horizon, D_{sense}. The robot is given a finite amount of time, τ_{plan}, within which it must find a plan, and it executes a duration $\tau_{\text{move}} \leq T$ of a given plan. Note that in real-world applications and previous sections of this paper, τ_{move} is the same as τ_{plan}. τ_{move} is defined separately in this section to simulate the RTD, RRT, and NMPC planners with and without real-world timing limits to compare performance. For the Segway and Rover, $\tau_{\text{move}} = 0.5$ s. Simulations are all implemented in MATLAB on a 2.10 GHz computer with 1.5 TB of RAM. Timeouts are enforced with MATLAB's `tic` and `toc` functions. For all planners and all simulations, since the robot is represented as the high-fidelity model (3.1), there is no state estimation error, so $\varepsilon = 0$ from Assumption 11.

For each robot, 1,000 random trials that fit the environments described in §5.1.1.3 and §5.1.2.3 are generated. Since these are randomly generated, it is not guaranteed that feasible (i.e. collision-free) paths exist from the start to the goal in every trial. This is useful, because it requires planners to be safe even when the high-level planner can only find infeasible paths to the goal.

This section is organized as follows. §5.1.4.1 describes Experiment 1, which is used to the set the buffer sizes for RRT and NMPC. §5.1.4.2 describes Experiment 2, which enforces realistic planning and sensing limits. §5.1.4.3 describes Experiment 3, which enforces the minimal sensor horizon according to Theorem 38. §5.1.4.4 concludes by summarizing the results for all experiments.

5.1.4.1 Experiment 1: Buffer Size for RRT and NMPC

The goal for Experiment 1 is to determine how to buffer obstacles for RRT and NMPC. To ensure that the buffer size is the only parameter that influences RRT and NMPC, this experiment relaxes the real-time and limited sensor horizon requirements, giving the planners enough time and information to find a plan in most planning iterations.

The parameters used for Experiment 1 are as follows. For the Segway, $\tau_{\text{move}} = 0.5$ s, $\tau_{\text{plan}} = 10$ s, and $D_{\text{sense}} = 100$ m. For the Rover, $\tau_{\text{move}} = 0.5$ s, $\tau_{\text{plan}} = 10$ s, and $D_{\text{sense}} = 30$ m. Since $\tau_{\text{plan}} > \tau_{\text{move}}$, the real-time requirement is relaxed. Since D_{sense} is large, the limited sensor horizon requirement is relaxed. For both the Segway and the Rover, obstacles are buffered by Minkowski sum with a polygonal outer approximation of a closed disk. For the Segway, since the robot's radius is 0.38 m, buffer sizes of 0.40, 0.45, and 0.50 m are tested. A buffer size of 0.65 m; is also tested which accounts for the braking distance of the Segway. For the Rover, obstacles are buffered in the (x, y) dimensions by a rectangle encompassing rotations of up to 0.6 rad (0.29, 0.26 m). Although a more complicated collision check could be used for the footprint, this type of buffering reduces

Segway Exp. 1		RRT		NMPC	
		Goals	Crashes	Goals	Crashes
Buffer [m]	0.40	83.6	3.6	86.2	11.7
	0.45	86.2	1.4	**97.0**	0.6
	0.50	81.9	0.4	96.0	0.4
	0.65	71.9	**0.0**	83.5	**0.0**

Table 5.1: Comparison of success and crash rates for varying buffer sizes for the Segway. A buffer size of 0.45 m provides the best balance of performance and safety for both RRT and NMPC. The Segway's braking distance of 0.625 m from 1.25 m/s means that the 0.65 m buffer prevents RRT and NMPC from crashing, but both methods become conservative with this buffer.

Rover Exp. 1		RRT		NMPC	
		Goals	Crashes	Goals	Crashes
Buffer [m]	0.29, 0.26	**99.8**	**0.0**	99.6	**0.0**
	0.34, 0.31	97.9	**0.0**	98.8	**0.0**
	0.39, 0.36	95.8	**0.0**	97.8	0.01

Table 5.2: Comparison of success and crash rates for varying buffer sizes for the Rover. Buffers are listed given in m in the (x, y) dimensions. A buffer size of (0.29, 0.26) maximizes performance without crashing.

computational complexity and is commonly used in driving applications [MUDL11]. Additional buffers of 0.0, 0.05, 0.10 are tested.

I expect the results of Experiment 1 to show that, as the buffer size is increased for both planners and both robots, the number of crashes reduces (because any plan that avoids a buffered obstacle places the robot farther away from the actual obstacle for a larger buffer size), and the number of goals reached reduces (because a larger buffer reduces the amount of free space available to each planner). I expect no crashes for either planner with a buffer size of 0.65 m for the Segway.

The results of Experiment 1 are summarized in Table 5.1 for the Segway and Table 5.2 for the Rover. Recall that, since the trials are randomly generated, every trial my not have a collision-free path from start to goal. On the Segway, RRT and NMPC fulfill the expectation that, as the buffer size increases, the number of goals and crashes both reduce; a buffer size of 0.45 m provies the best balance between goals and crashes. On the Rover, that the buffer size of the expanded footprint plus 0.0 m has the best performance with no crashes. Surprisingly, NMPC had a crash with the largest buffer size for the Rover. In this instance, the solver was unable to find a feasible solution in one planning iteration because too much free space was removed due to the buffered obstacles. This resulted in the robot colliding with the simulated environment boundary after while trying to emergency brake.

Crashes occur for the RRT and NMPC planners because the smaller buffer sizes are potentially

too small to compensate for both robots' inability to perfectly track a planned trajectory (recall that, in this implementation, RRT plans trajectories with an RK4 or forward Euler approximation of the high-fidelity model, and NMPC uses a polynomial approximation). This is addressed in subsequent experiments by choosing an RRT and NMPC buffer size of 0.45 m for the Segway and $(0.29, 0.26)$ m for the Rover. This choice is a balance between a high success rate and a low crash rate.

RRT and NMPC are also tested on the Segway with buffer sizes of 0.65 m, to check that, if no feasible solution is found in a planning iteration, both methods should always be able to brake without crashing. The largest buffer size results in the most conservative performance, with 71.8% of goals reached for RRT, and 83.5% for NMPC. As expected, both planners are always able to come to a stop without crashing.

5.1.4.2 Experiment 2: Real-time Planning and Limited Sensor Horizon

The goal for Experiment 2 is to understand the performance of RTD, RRT, and NMPC when subject to real-time and limited sensor horizon requirements. RTD is designed to satisfy these requirements while provably ensuring safety. RRTs are typically capable of rapid planning, though not necessarily with arbitrary dynamics [ES14, KTF+09]. For NMPC, these requirements can cause wide variations in performance depending on how constraints are represented [FGZ+13, GGC+14, PR14, HK07, UAB+08].

The parameters used for Experiment 2 are as follows. For the Segway, $\tau_{\text{move}} = \tau_{\text{plan}} = 0.5$ s and $D_{\text{sense}} = 4.0$ m. For the Rover, $\tau_{\text{move}} = \tau_{\text{plan}} = 0.5$ s, and $D_{\text{sense}} = 5$ m. Since $\tau_{\text{move}} = \tau_{\text{plan}}$, the amount of time allowed for planning is the same as the amount of time that each robot executes from the previously-planned trajectory, meaning the real-time requirement is enforced. Since D_{sense} is smaller than the size of each robot's environment (see §5.1.1.3 and §5.1.2.3), the limited sensor horizon requirement is enforced. The RRT and NMPC buffer size is 0.45 m for the Segway, and (0.29, 0.26) m for the Rover. The buffer sizes used for RTD are given in §5.1.1.2 and §5.1.2.2.

I expect the results of Experiment 2 to be as follows. For both robots, I expect RTD to have a similar number of goals reached as RRT and NMPC, and I expect RRT and NMPC to reach the goal less often than in Experiment 1. This is due to the limited sensor horizon, meaning the high-level planner no longer has access to the entire environment at time 0, and therefore may make poor routing decisions. As for crashes, RTD is designed with real-time performance as a requirement, and prescribes a minimum sensor horizon in Theorem 38 that is less than D_{sense} for both robots. Therefore, I expect RTD to have no crashes. I expect RRT and NMPC to have slightly more crashes than in Experiment 1, because the sensor horizon is shorter, and because the real-time requirement means that these two planners may be unable to find feasible plans as often, resulting in both planners braking more frequently.

The results of Experiment 2 are summarized in Tables 5.3 for the Segway and Table 5.4 for the Rover. For the Segway, RTD reaches the goal more often than the other two planners do in Experiment 1 or in Experiment 2 (96.3%); recall that the same environments are used in all three experiments, making this comparison possible. RRT surprisingly reaches the goal less often in Experiment 2 than in Experiment 1 (78.2% vs . 86.3%); and NMPC is incapable of reaching the goal (0% vs. 83.7%). RTD has no crashes, as expected; RRT crashes less often (2.4% vs. 3.6%); and NMPC does not crash because it struggles to move the robot at all. For the Rover, RTD reaches the goal 95.4% of the time. RRT reaches the goal slightly less often than in Experiment 1 (97.6% vs. 99.8%); and NMPC is incapable of reaching the goal (0% vs. 99.6%). RTD has no crashes; RRT crashes once (0.01%); and NMPC does not crash because it struggles to move the robot.

For both the Segway and Rover, RTD's performance is as expected based on the theory in Chapter 4: it is able to reach the goal, can plan in real time, and has no crashes. The Segway's RRT has a reduction in crashes, which is surprising, but is likely because the real-time requirement means that RRT is less likely to find a feasible plan at every iteration, and must brake more often. For the Segway's NMPC planner, GPOPS-II is able to find trajectories rapidly when the vehicle is not near obstacles; but, since the obstacles are randomly-placed and produce non-convex constraints, the solver struggles to solve quickly when near them, resulting in 0 goals and 0 crashes. For the Rover, compared to Experiment 1, the RRT planner reaches the goal slightly less often, but still crashes, as expected due to the reduced planning time limit; unlike the Segway, the Rover cannot spin in place to potentially find a new plan after braking. The Rover's NMPC planner suffers the same issues near the obstacle constraints as the Segway's NMPC planner. It is worth noting that, for the Rover, I was able to generate heuristics for the RRT that exploited the structure of the environment, which enables the RRT to more goals than RTD. However, in the random environments generated for the Segway, RTD reaches more goals than RRT.

Figure 5.1 demonstrates Experiments 1 and 2 for the Segway; the RRT and NMPC plots are from Experiment 1, and the RTD plots are from Experiment 2, since RTD is not run in Experiment 1. The figure shows one environment where RTD, RRT, and NMPC all reach the goal without crashing; one environment where RTD reaches the goal, RRT crashes, and NMPC gets stuck; and one environment where RTD brakes safely whereas RRT and NMPC reach the goal. In the second environment, RRT crashes because, while trying to navigate a gap between two obstacles, it is unable to find a feasible plan; it then attempts to brake along its previous trajectory, but touches an obstacle while doing so. NMPC gets stuck trying to navigate this same gap where RRT crashes, because the gap is a non-convex region with enough obstacle constraints that the NMPC planner computes slowly. Unlike RRT, NMPC brakes much earlier, but then is unable to find a plan to navigate the gap. In the third environment, RTD gets stuck because, early on, it finds a different path from RRT and NMPC; this new path causes the high-level planner to reroute RTD towards a

region where the high-level planner believes that the route is feasible, but RTD determines that it is not, resulting in RTD braking safely. This demonstrates that, even if the high-level planner makes infeasible decisions, RTD is safe.

Figure 5.2 demonstrates Experiment 2 for the Rover with one environment where RTD succeeds, RRT crashes, and NMPC gets stuck; and one environment where all planners brake safely. RRT crashes when it travels too close to an obstacle to find a feasible plan at the next planning iteration, causing it to try to brake, resulting in a crash. In some environments, NMPC is able to find plans until the obstacles appear in its sensor horizon.

5.1.4.3 Experiment 3: Real Planning Time and Minimal Sensor Horizon

The goal for Experiment 3 is to confirm that RTD performs safe, real-time trajectory planning even when the sensor horizon is the minimum possible as per Theorem 38. This is useful because, to be practical, RTD must be able to tolerate environments where a robot's sensors are only effective in a small area.

The parameters used for Experiment 3 are as follows. For the Segway, $\tau_{\text{move}} = \tau_{\text{plan}} = 0.5$ s and $D_{\text{sense}} = 1.9$ m. For the Rover, $\tau_{\text{move}} = \tau_{\text{plan}} = 0.5$ s and $D_{\text{sense}} = 4$ m. Since $\tau_{\text{move}} = \tau_{\text{plan}}$, the real-time planning requirement is enforced, as in Experiment 2. The sensor horizon D_{sense} is given by Theorem 38, assuming the distance is bounded by the robot's max speed. Buffer sizes for both robots are the same as in Experiment 2.

I expect the results of Experiment 3 to show that RTD has zero crashes for either robot. I expect the number of goals reached to be less than those in Experiment 2, because a smaller sensor horizon means that the high-level planner for both robots has less information when making routing decisions. So, there may be more environments where the high-level planners cause both robots to brake safely without reaching the goal.

The results of Experiment 3 confirm the expectation. Both robots have 0 crashes. The Segway reaches the goal 96.2% of the time, versus 96.3% in Experiment 2. The Rover reaches the goal 95.2% of the time, versus 95.4% in Experiment 2. Neither robot has any crashes with the minimal sensor horizon. Furthermore RTD maintains performance in terms of goals reached; this is likely because when the distance bound D_{max}, presented in Theorem 38, conservatively assumes the robot is traveling at its maximum speed. This means, intuitively, that a smaller sensor horizon is sufficient at lower speeds.

5.1.4.4 Discussion

The experiments, summarized in Tables 5.3 and 5.4, show that RTD is successful in reaching the desired goal comparably often to RRT and NMPC for both the Segway and Rover. Importantly,

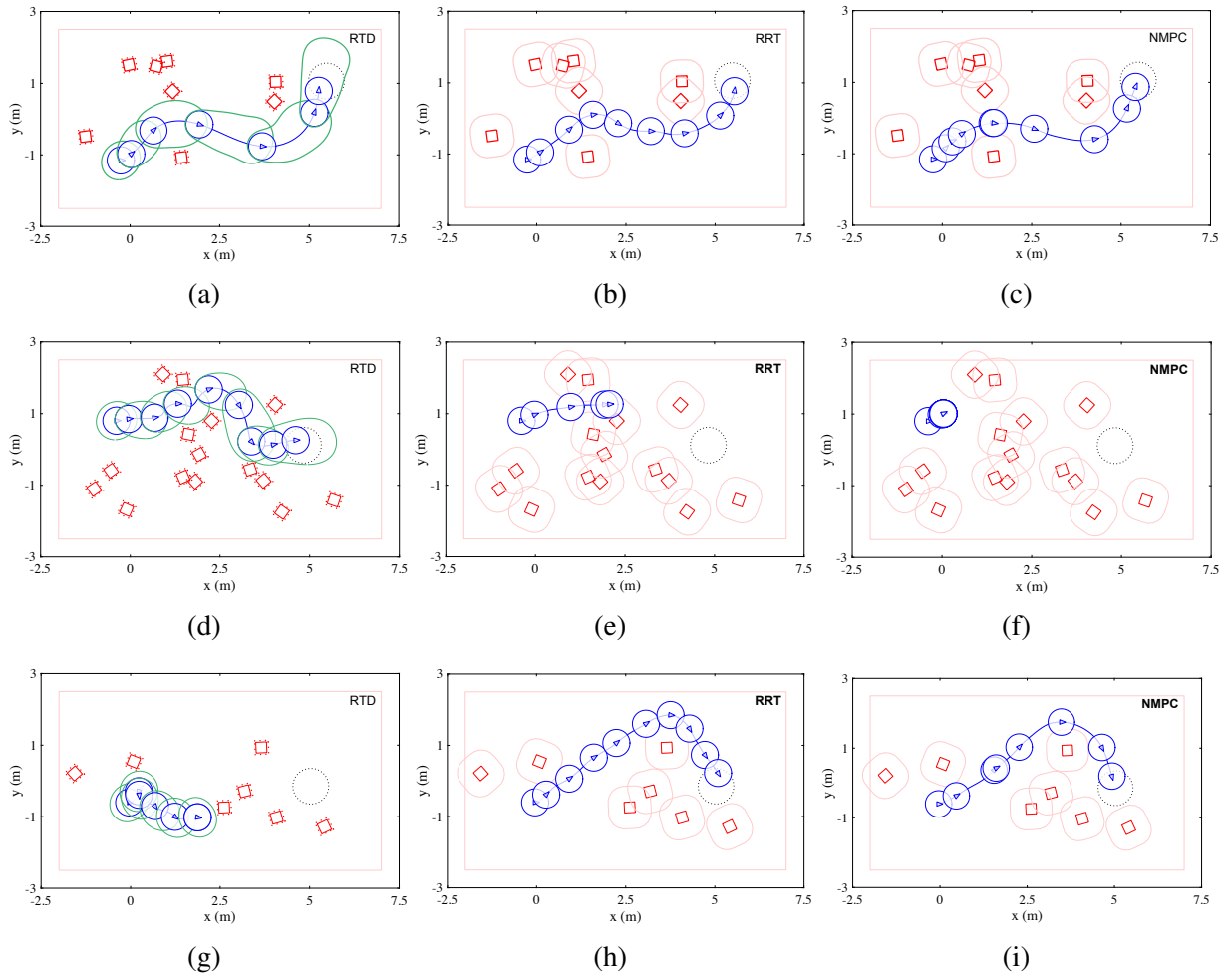


Figure 5.1: Sample environments from Experiment 2 for the Segway, which starts on the west (left) side of the environment, with the goal plotted as a dotted circle on the east (right) side of the environment. The Segway's pose is plotted as a solid circle every 1.5 s, or less frequently when the Segway is stopped or spinning in place. For RTD, contours of the FRS (i.e. the edge of $\pi_X(k^*)$ (4.14)) are plotted. The obstacles for all three planners are plotted as solid boxes. For RTD, the discretized obstacle is plotted as points around each box. For RRT and NMPC, the buffered obstacles are plotted as light lines around each box. Row 1 (Subfigures (a), (b), and (c)) shows an environment where all three planners are successful. Row 2 shows an environment where RTD is successful, but RRT and NMPC are not. Subfigure (d) shows RTD reaching the goal. Subfigure (e) shows RRT attempting to navigate a gap between several obstacles, where it is unable to find a new plan; it crashes when it tries to brake. Subfigure (f) shows NMPC braking because it cannot compute a safe plan to navigate the same gap; here, NMPC brakes safely and gets stuck. Row 3 shows an environment where RTD fails to reach the goal, but RRT and NMPC do. Subfigure (g) shows that RTD initially turns north more sharply than RRT or NMPC, which forces it to brake safely; but that causes the high-level planner to reroute it south, where there ends up being no feasible solution. Subfigures (h) and (i) show RRT and NMPC reaching the goal because they do not turn north as sharply initially, and the high-level planner routes them around the obstacles.

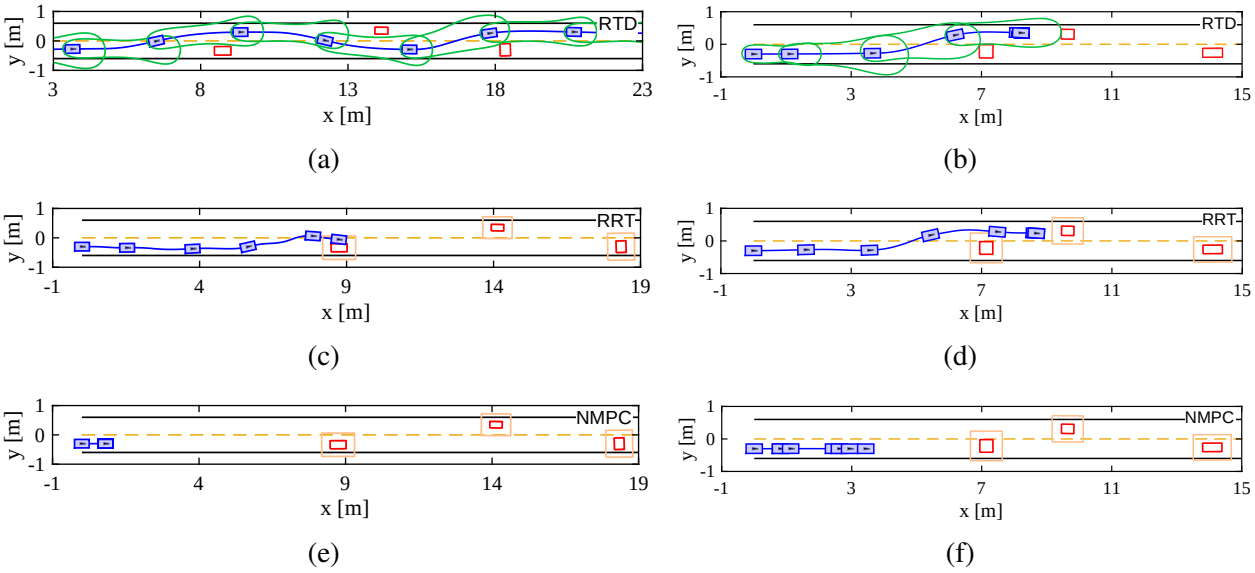


Figure 5.2: Two sample environments from Experiment 2 for the Rover. The Rover's trajectory, starting from the far left, is a solid line, and its pose at several sample time instances is plotted with solid rectangles. Obstacles are plotted as red boxes. Buffered obstacles for RRT and NMPC are plotted with light solid lines. Subfigures (a) and (b) show RTD avoiding the obstacles. The subset of the FRS associated with the optimal parameter every 1.5 s is plotted as a contour. Subfigures (c) and (d) show the RRT method. In Subfigure (c), RRT is unable to safely track its planned trajectory around the first obstacle. In Subfigure (d), RRT is able to come to a stop before the second obstacle. Subfigures (e) and (f) show NMPC, which stops due to enforcement of real-time planning limits.

Segway Simulation Results					
Experiment	τ_{plan} [s]	D_{sense} [m]	Planner	Goals [%]	Crashes [%]
1	10.0	100	RRT	86.2	3.6
			NMPC	**97.0**	**0.6**
2	0.5	4.0	RTD	**96.3**	**0.0**
			RRT	78.2	2.4
			NMPC	0.0	**0.0**
3	0.5	1.5	RTD	96.2	**0.0**

Table 5.3: Simulation results of Experiments 1–3 for the Segway. RTD is the only method that never experiences crashes, as expected; it also reaches the goal more frequently than RRT or NMPC. NMPC reaches the goal more often than RTD and RRT, with fewer crashes than RRT, but is unable to plan in real time (Experiment 2). In Experiment 3, RTD is capable of planning safely when given the smallest possible sensor horizon allowed for by Theorem 38.

Rover Simulation Results					
Experiment	τ_{plan} [s]	D_{sense} [m]	Planner	Goals [%]	Crashes [%]
1	10.0	30	RRT	**99.8**	**0.0**
			NMPC	99.6	**0.0**
2	0.5	5.0	RTD	95.4	**0.0**
			RRT	**97.6**	0.1
			NMPC	0.0	**0.0**
3	0.5	4.0	RTD	95.2	**0.0**

Table 5.4: Simulation results of Experiments 1–3 for the Rover. RTD is the only method that can both reach the goal and never crash when real time planning is enforced. In the Rover's road-like environment, RRT has excellent performance, but crashed in 1 out of 1000 trials when the real-time planning limit was enforced. In Experiment 3, RTD is capable of planning safely when given the smallest possible sensor horizon allowed for by Theorem 38.

RTD has 0 crashes in all of the simulations. RRT crashes because its paths may take it near obstacles, where it is difficult to build a dense tree since most nodes are infeasible. When this happens, RRT attempts to brake, but there is no guarantee that this can be done safely. Interestingly, for the Segway, reducing the allowed planning time τ_{plan} reduces the crash rate. This is because RRT cannot find a feasible plan as frequently with the lower planning time, so it brakes more often, and begins braking when further away from obstacles. NMPC crashes because, when the robot is near an obstacle, there are a large number of non-convex constraints in the resulting optimization program, so finding a feasible solution within the planning time τ_{plan} is difficult. If no plan is found, the robot attempts to continue executing its last feasible plan (which includes a braking maneuver), but the algorithm has no guarantee that doing so is safe. Increasing the buffer size (Experiment 1)

reduces the number of crashes for both RRT and NMPC for the Segway, as expected. The tradeoff for buffer size is that a larger buffer reduces the free space available for the robot to move through, reducing how often each robot reaches the goal. Importantly, crashes occur for both planners even when they are not required to plan in real-time or with a limited sensor horizon.

RTD is sometimes unable to reach the goal, but still always brakes safely. Note that RRT and NMPC on both the Segway and Rover platforms are sometimes also unable to reach the goal. For the Segway, stopping safely before reaching the goal occurs when RTD plans a path too close to an obstacle, in which case the online optimization struggles to find a non-stopped solution even after spinning the Segway in place. This may be remedied by changing the high-level planner to penalize obstacles more, or by changing the cost function in the online optimization. The Rover stops without reaching the goal when the reachable set is too large to make a lane change through a tight gap between two obstacles. This may be due to the fact that the decomposition technique used to compute the FRS's is conservative when the footprint rotates. This could be remedied by using a simpler trajectory parameterization, like the Segway's, in low-speed, tight scenarios.

For the Rover's environments RRT and NMPC have excellent performance in Experiment 1. The sparse (compared to the Segway), structured, and static environment, eases the development of heuristics for both the waypoint and trajectory planners. The benefits of RTD are greater in the random environments generated for the Segway. See Figures 5.1 and 5.2 for examples of RTD performing trajectory planning for the Segway and Rover platforms.

5.1.5 Hardware Demonstration

This section details the application of RTD to the Segway (Figure 3.2a) and Rover (Figure 3.2b) hardware platforms. The hardware demonstrations affirm this point. Videos of the robots are available at `https://youtu.be/FJns7YpdMXQ` for the Segway and `https://youtu.be/bgDEAi_Ewfw` for the Rover.

5.1.5.1 Segway

The first hardware demo uses the Segway Robotics Mobility Platform shown in Figure 3.2a. Sensing is performed with a Hokuyo UTM-30LX planar lidar; in practice, this sensor is accurate up to $D_{\text{sense}} = 4.0$ m away (recall that the Segway runs indoors, so the effective sensor horizon is small). The robot is controlled by a 4.0 GHz laptop with 64 GB of memory, running MATLAB and Robot Operating System (ROS). Google Cartographer is used for localization and mapping [HKRA16]. All computation is run onboard. Since SLAM and state estimation requires 0.2 s per iteration, τ_{plan} is set to 0.3 s when calling `fmincon` for trajectory optimization in Algorithm 1 Line 7. In practice that the state estimation error is never more than 0.1 m in the global xy-coordinate frame while the

Segway predicts its future state (Algorithm 1 Line 9), so for Assumption 11, $\varepsilon_x = \varepsilon_y = 0.1$ m. The FRS is computed for the Segway as described in §5.1.1.2.

The Segway is run on a 4×8 m^2 tile floor with 30 cm cubical obstacles randomly distributed just before run time. The Segway has no prior knowledge of the obstacles. Two points are picked on opposite ends of the room and used as the start and goal points in an alternating fashion.

A supplementary video illustrates the performance of RTD. Despite the randomly-placed obstacles, the Segway RMP platform is able to operate safely while consistently reaching its goal. As in the simulation, the Segway uses a low speed and a high speed FRS. In the handful of instances where the Segway brakes, the high-level planner generates waypoints that require passing through a gap that is too small for the high speed FRS; the Segway swaps to the low speed FRS after stopping, and is then able to navigate the gap.

5.1.5.2 Rover

The second hardware demo uses a Rover car-like robot based on a Traxxas RC platform. The Rover is tested on a 7 m long mock road, which is a tiled surface, as shown in Figure 3.2b. This setup resembles the simulation environment, but with a shorter road and smaller obstacles. The Rover is equipped with a front-mounted Hokuyo UST-10LX planar lidar for sensing and localization; as the Rover runs indoors, this sensor is accurate up to at least $D_{\text{sense}} = 3.5$ m away given occlusions and obstacle density. An NVIDIA TX-1 computer on-board is used to run the sensor drivers, state estimator, feedback controller, and low-level motor controller. The Rover uses ROS to communicate with an Intel Core i7 7820HK (2.90 GHz) CPU/64 GB RAM laptop over wifi. The laptop is used for localization and mapping, to capture experiment data, and to run `fmincon` for trajectory optimization in Algorithm 1 Line 7. The state estimation error in Assumption 11 is bounded by $\varepsilon_x = \varepsilon_y = 0.1$ m. The FRS is computed for the Rover as described in Section 5.1.2.2.

For each trial, the Rover is placed at one end of the mock road and instructed to drive to a goal at the other end at speeds of 1–1.5 m/s. One to three obstacles are placed between the Rover and the goal. The obstacles are $0.3 \times 0.3 \times 0.3$ m^3 cardboard cubes. The Rover is not given prior knowledge of the obstacles for each trial, and uses its planar lidar to detect them in real-time. The Rover has an enforced planning time limit of $\tau_{\text{plan}} = 0.375$ s. Contrary to the Segway, the mapping did not take a significant amount of the planning time. This is because localization and map updates were provided smoothly at 20 Hz, so the algorithm did not need to pause and wait for an update as often as the Segway did. Eight trials were run back-to-back and filmed in one take, as presented in the supplementary video. Several types of scenarios are constructed to encourage the Rover to change lanes or force it to brake to a stop. Eighteen trials were run in addition to the filmed trials, and resulted in zero crashes. The Rover uses one FRS to plan at speeds between 1.0–1.5 m/s. Due to the minimum speed, the Rover is occasionally unable to navigate tight gaps;

this could be remedied by using a low speed FRS with a different trajectory parameterization.

5.2 CarSim Implementation

This section describes the application of RTD to a CarSim model of a Ford Fusion [VSK+19]. The dynamic models used for prediction and for planning are discussed in §5.2.1. The FRS computation and online planning implementation are presented in §5.2.2. A simulation comparison on a test track against RRT and NMPC planners with static obstacles is presented in §5.2.5. A video of RTD navigating a sample track and performing emergency braking can be found at `https://youtu.be/lmtki6elFlw`.

5.2.1 Platform

This paper implements RTD on a passenger car model of a Ford Fusion in CarSim. The inputs are throttle, steering wheel angle, and brake master cylinder pressure. *vehicle* to refers to the Carsim model in this section.

A bicycle model similar to [LDM15, (1)] is used as the high-fidelity model (3.1):

$$\frac{d}{dt}\begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \\ v_x(t) \\ v_y(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} v_x(t)\cos\theta(t) - v_y(t)\sin\theta(t) \\ v_x(t)\sin\theta(t) + v_y(t)\cos\theta(t) \\ \dot{\theta}(t) \\ \frac{1}{m}F_x(z_{\text{hi}}(t)) - \frac{1}{m}F_{y,f}(\alpha_f(z_{\text{hi}}(t)))\sin(\delta(t)) + v_y(t)\dot{\theta}(t) \\ \frac{1}{m}F_{y,f}(\alpha_f(z_{\text{hi}}(t)))\cos(\delta(t)) + \frac{1}{m}F_{y,r}(\alpha_r(z_{\text{hi}}(t))) - v_x(t)\dot{\theta}(t) \\ \frac{l_f}{I}F_{y,f}(\alpha_f(z_{\text{hi}}(t)))\cos(\delta(t)) - \frac{l_r}{I}F_{y,r}(\alpha_r(z_{\text{hi}}(t))) \end{bmatrix}, \tag{5.4}$$

where x and y are position; θ is the vehicle's heading in the global coordinate frame; v_x, v_y are longitudinal and lateral speed of the center of mass; and $\dot{\theta}$ is yaw rate. The constants m, I, l_f, and l_r are the vehicle's mass, yaw moment of inertia, distance from the front wheel to center of mass, and distance of the rear wheel to center of mass. I fit polynomials relating the inputs to the driving force, F_x, and find a linear relationship between wheel angle, δ, and steering wheel angle. I fit a simplified Pajecka tire model [LDM15, (2a, 2b)] to the lateral tire forces, F_y as functions of the slip angles, α. Since F_x, $F_{y,f}$, and $F_{y,r}$ are continuous, the dynamics (5.4) are continuous and Assumption 2 is satisfied.

5.2.2 Forward Reachable Set Computation

Recall that (5.4) cannot perfectly capture the motion of the vehicle, however Algorithm 1 must be able to predict the vehicle's future position 1 planning iteration in the future. The high-fidelity model is simulated and compared to Carsim data to empirically find the error bounds for Assumption 11: $|\varepsilon| \leq [0.1,\ 0.1,\ 0.12,\ 0.15,\ 0.02,\ 0.4,\ 0.08,\ 0.05]^\top$ where $|\cdot|$ is taken element-wise.

The trajectory-producing model (3.2) has dynamics:

$$f(t, z(t), k) = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \end{bmatrix} = \begin{bmatrix} k_1 - k_2 y(t) \\ v_y + k_2 x(t) \end{bmatrix} \tag{5.5}$$

$$v_y = k_2 \left(l_r - \frac{m\, l_f}{C_{y,r}\,(l_r + l_f)} k_1^2 \right), \tag{5.6}$$

where $z = [x, y]^\top$; k_1 (resp. k_2) specifies a constant desired speed (resp. yaw rate) and $C_{y,r}$ is the rear cornering stiffness from the tire force model in (5.4). The lateral speed, v_y, is derived from steady-state, linear tire force assumptions [SHB14, Section 10.1.2]. Notice that similar to Example 5, (3.2) only has the two states x and y, i.e. heading dynamics are omitted. Consequently, (5.5) produces trajectories of the vehicle's entire footprint in X. For any $k \in K$, the high-fidelity model generates a feedback controller $u_k : [0, T] \times Z_{\text{hi}} \to U$ to track the trajectory parameterized by k. For this platform, u_k is implemented with linear MPC in MATLAB.

now discuss tacking error and the FRS computation. In this work, g_x and g_y are polynomials of degree 2 that overapproximate tracking error data as in Assumption 7. Importantly, state estimation error in $z_{\text{hi},0}$ is added to the initial conditions, so g conservatively approximates dynamics associated the prediction errors described in Assumption 11. The FRSs are computed using the SOS implementation of (D) from §3.3.2 with $l = 5$. The change in velocity and yaw rate is limited to 1 m/s and 0.25 rad/s between planning iterations. Figure 5.3 shows tracking error data and g. I fit the g function using (5.4), but verify that it is conservative with respect to data from the real vehicle, the CarSim model, as described in Remark 8.

5.2.3 Environment

The vehicle runs on a 1036 m, counter-clockwise, closed loop test track with 7 turns (with approximate curvatures of 0.005–0.04 m^{-1}) and two 4 m wide lanes. Twenty stationary obstacles (with random length of 3.3–5.1 m length and width of 1.7–2.5 m) are distributed around the track in random lanes and randomly spaced 40–55 m apart. Ten random tracks were generated; though the mean obstacle spacing is the same, the tracks vary in difficulty. For example, some tracks require performing overtaking maneuvers in a corner. The vehicle begins each simulation at the northwest corner of the track in the left lane, with first obstacle at least 50 m away. A high-level planner

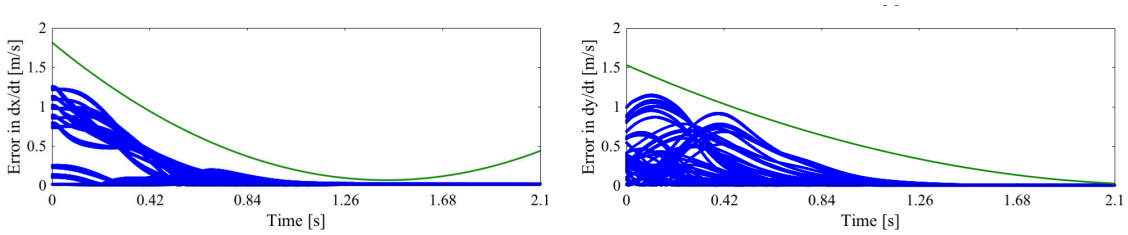


Figure 5.3: Example of tracking error in x and y plotted for a reference trajectory of $k_1 = 12$ m/s, $k_2 = 0$ rad/s, and $T = 2.1$ s. Data (blue) is from CarSim and captures initial velocities and yaw rates between 10.78 to 13.26 ms and -0.25 to 0.25 rad/s. The green lines is the error functions g_x and g_y as in (3.12).

places waypoints ahead of the vehicle at a lookahead distance proportional to the vehicle's current speed. If the lane centerline from the vehicle's current position and lane to the waypoint intersects an obstacle, the waypoint is switched to the other lane to encourage a lane change. Lane keeping is not explicitly enforced but is encouraged via the cost function. A trial is successful if the vehicle completes one lap of the track.

5.2.4 Trajectory Planner Implementations

This section describes implementation aspects of the RTD, Rapidly-Exploring Random Tree (RRT) and Nonlinear Model-Predictive Control (NMPC) trajectory planners. A planning time limit of $\tau_{\text{plan}} = 0.5$ is enforced for all planners, RRT and NMPC planners are also tested with limits of 10 s.

5.2.4.1 RTD

For RTD, the discrete obstacle representation in §4.2.2 is used with quantities $b = 0.05$ m, so $r = 0.1$ m and $a = 0.07$ m To satisfy Theorem 38, the vehicle has a minimum sensor horizon of $D_{\text{sense}} = 42.4$ m as in Assumption 30; which is well within the range specified by commercial LIDAR units [Aco07]. The trajectory optimization program in Algorithm 1 Line 7 is solved with `fmincon`. If there is no feasible solution, the vehicle brakes along the previously planned path. I empirically verified that braking in this manner stays within the FRS; however, the more appropriate approach is to compute an FRS with time-varying dynamics as described in §3.5. To keep the vehicle on the road, RTD buffers the road boundaries by 2.5 m outside the road and incorporates these buffers as obstacles. Since the FRS includes the full vehicle body, this ensures the vehicle's center of mass stays on the road when tracking any trajectory planned by RTD. Figure 5.4 shows an timelapse of RTD navigating a section of the track.

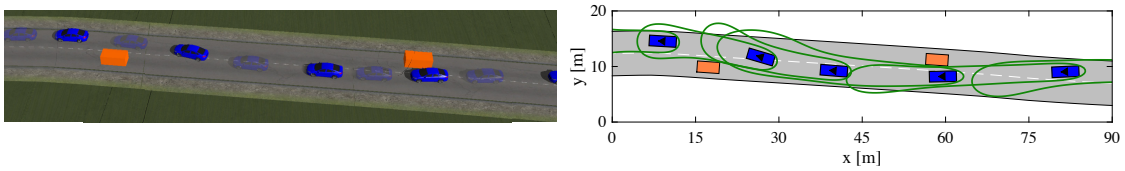


Figure 5.4: The vehicle (solid blue) autonomously performs lane change maneuvers at 8–15 m/s around obstacles (orange) on a 90 m section of road, beginning from the right side of the figure. The presented RTD method drives the vehicle safely in real time. The left plot shows the vehicle in Carsim, with the vehicle transparent at intermediate times to show motion. The right plot shows the RTD planner in MATLAB, with the green contours in showing the forward reachable set at each planning iteration. A video is available at `https://youtu.be/lmtki6elFlw`.

5.2.4.2 RRT

The RRT planner is implemented based on [KTF+09] and similar to §5.1.3.2. To compensate for the vehicle's footprint, obstacles are buffered by 4 m in length and 1.5 m in width. Nodes are created by forward-integrating (3.1) from a randomly-selected node with randomly-chosen control inputs held for 0.5 s to create 50 points spaced 0.01 s apart; the last such point is the new node, which is discarded if any of the points leave the track or enter a buffered obstacle. Two trees are built in parallel: one with throttle inputs, and one with braking inputs. The cost at each node is distance to the current waypoint, plus penalties for being near obstacles or road boundaries, and for commanding large control inputs.

5.2.4.3 NMPC

The NMPC planner uses GPOPS-II, a commercially available pseudo-spectral nonlinear MPC solver [PR14], with a kinematic bicycle model, similar to §5.1.3.3. The inputs are acceleration and steering wheel angle rate; this model was used to reduce solve time, and has been shown to effective for trajectory planning in mild to moderate conditions [PAdNdLF17]. Obstacles are buffered by 4 m in length and 1.25 m in width. The track is represented as a set of adjacent rectangles. Constraints are created as half-planes to ensure that the planned trajectory (of the center of mass) does not enter buffered obstacles or exit the track.

5.2.5 Simulation Results

The simulations are run on a 2.6 GHz computer with 128 GB RAM. Planning times are reported using Matlab's `tic` and `toc` functions. All planners use a receding horizon strategy with $\tau_{\text{plan}} = 0.5$ s. In the first experiment, all three planners are run with a real-time planning limit enforced. In the second experiment, RRT and NMPC are given extra time. Results are shown in Table 5.5.

Planner	Planning Time (s)		% of Track Complete		Crashes	Safe Stops
	Avg	Max	Avg	Max		
RTD	**0.09**	0.50	**100**	100	0	0
RRT	5.00	5.00	31	86	0	10
	0.50	0.50	13	38	1	9
GPOPS	3.71	72.58	**100**	100	0	0
	0.50	0.50	0	0	0	10

Table 5.5: Simulation results comparing RTD, RRT [KTF+09], and GPOPS-II [PR14] on 10 simulated tracks. The experiment with the real-time planning limit is shown in gray; the one without is shown in white. The fourth and fifth columns show the average and max percent of each track completed. The six and seventh columns count the number of crashes or safe stops if the vehicle did not complete the track.

RTD successfully navigates the track in all 10 trials, with an average planning time of 0.086 s. In the first experiment (enforcing real-time planning) RRT navigates 13% of the track on average. When the vehicle approaches obstacles, the planner struggles to generate feasible nodes that avoid the obstacle while staying on the track. Since the algorithm attempts to plan a braking trajectory at each iteration, it is able to stop safely (without colliding with an obstacle) in 9 trials; it has 1 crash because it cannot always plan a feasible braking trajectory. Increasing the buffer size of the obstacles could reduce collisions, but would impact performance. GPOPS-II is unable to plan trajectories in less than 0.5 seconds due to the number of track constraints; hence, it records 10 safe stops. In the second experiment, the extended planning time allows RRT to complete more of the track. Although unable to always reach the goal, RRT uses the extended planning time to find feasible braking plans. GPOPS-II successfully reaches the goal in all 10 trials, but with an average planning time of 3.71 s. The planning times for GPOPS-II have a standard deviation of 4.20 s; the large standard deviation is expected because the number of constraints vary based on the track curvature. Heuristics may reduce the amount of constraints, but would be obstacle- or track-specific. In contrast, the average planning time and standard deviation of RTD is 0.09 s and 0.06 s, so I do not expect its performance to vary if the track changes.

5.3 Dynamic Obstacles

This section describes the application of RTD to to environments with dynamic obstacles. Two platforms, the Segway from Figure 3.2a and a small electric vehicle pictured in Figure 5.5 are used. The discrete time collision check §4.3.1 [VKL+19] and the interval time collision check §4.3.2 [VLK+19] are both implemented and compared to the state-of-the-art. §5.3.1 describes the Segway and EV platforms. §5.3.2 describes the FRS computation for the Segway and EV

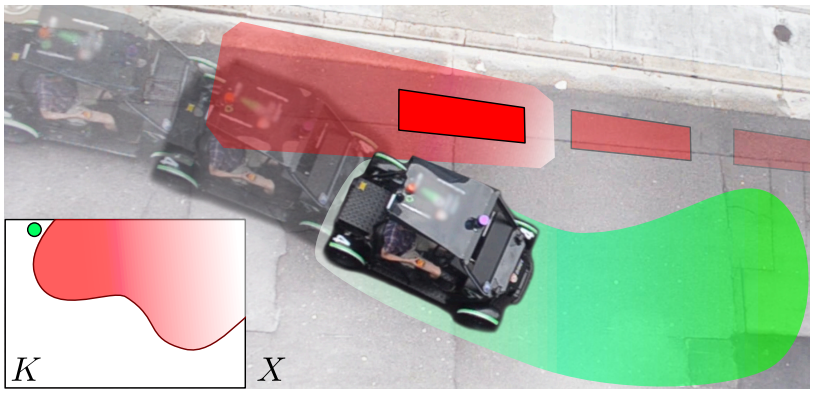


Figure 5.5: Depiction of RTD planning a trajectory for the EV robot (moving from left to right) around a dynamic obstacle (in red, moving from right to left) in the plane X. Opacity increases with time. At the last depicted time instance, the obstacle's predicted motion fades from white to red, and the forward reachable set of the EV fades from white to green. In the trajectory parameter space K, the planned trajectory is a green point lying outside the parameters for which the robot could be at-fault in a collision.

platforms. §5.3.3 describes the simulation environments they operate in. §5.3.5 describes the simulation results. §5.3.6 describes the hardware demonstrations.

5.3.1 Platforms

This section describes the platforms and dynamic models used. The first robot is the differential-drive Segway RMP with a high-fidelity model from Example 4 and §5.1.5.1. The second robot is the small Electric Vehicle (EV) in Figure 5.5 with the following high-fidelity model:

$$\frac{d}{dt}\begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \\ v(t) \\ \delta(t) \end{bmatrix} = \begin{bmatrix} v(t)\cos(\theta(t)) - \dot{\theta}(t)(c_1 + c_2 v(t)^2)\sin(\theta(t)) \\ v(t)\sin(\theta(t)) + \dot{\theta}(t)(c_1 + c_2 v(t)^2)\cos(\theta(t)) \\ \tan(\delta(t))v(t)(c_3 + c_4 v(t)^2)^{-1} \\ c_6 + c_7(v(t) - u_2(t)) + c_8(v(t) - u_2(t))^2 \\ c_5(\delta(t) - u_1(t)) \end{bmatrix}, \tag{5.7}$$

where θ is heading, δ is steering angle, and v is speed. Saturation limits are $|\delta(t)| \leq 0.50$ rad, $|\dot{\delta}(t)| \leq 0.50$ rad/s, and $|\dot{v}(t)| \in [-6.86, 3.50]$ m/s^2. For Assumption 11 $\varepsilon_x = 0.1$ m and the coefficients $c_1, \ldots, c_8$ were fit using localization data; the EV performs localization with a Robosense RS-Lidar-32 and saved maps [BWWN18]. The EV has a rectangular 2.4×1.3 m^2 footprint. ROS runs on-board on a 2.6 GHz computer. RTD is run in MATLAB on a 3.1 GHz laptop.

5.3.2 Forward Reachable Set Computation

For RTD, both robots create desired trajectories with the time-switching model:

$$f(t, z(t), k) = \begin{cases} \begin{bmatrix} k_1 - \dot{\theta}_{\text{des}}(k) y(t) \\ \dot{\theta}_{\text{des}}(k) x(t) \end{bmatrix}, & t \in [0, \tau_{\text{plan}}] \\ s(t,k) \begin{bmatrix} k_1 - \dot{\theta}_{\text{des}}(k) y(t) \\ \dot{\theta}_{\text{des}}(k) x(t) \end{bmatrix}, & t \in [\tau_{\text{plan}}, \tau_{\text{plan}} + \tau_{\text{brake}}(k)] \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, & t \in [\tau_{\text{plan}} + \tau_{\text{brake}}(k), T] \end{cases} \tag{5.8}$$

This model produces circular arcs that brake to a stop over $[\tau_{\text{plan}}, T]$. The trajectory parameters are $k = (k_1, k_2)$. The desired yaw rate is $\dot{\theta}_{\text{des}} : K \to \mathbb{R}$, given by $\dot{\theta}_{\text{des}}(k) = k_2$ for the Segway and $\dot{\theta}_{\text{des}}(k) = k_1 k_2 / l$ for the EV, where l is the EV's wheelbase in meters. For both robots, k_1 is desired speed. The braking time is $\tau_{\text{brake}}(k) = 1.0$ s for the Segway and $\tau_{\text{brake}}(k) = k_1/3$ for the EV. T is picked by sampling braking time for the high-fidelity models. The function $s : [0, T] \times K \to \mathbb{R}$ is given by

$$s(t, k) = 1 - \frac{t - \tau_{\text{plan}}}{\tau_{\text{brake}}(k)}. \tag{5.9}$$

For the Segway, $k_1 \in [0, 2]$ m/s and $|k_2| \leq 1.5$ rad/s ; between planning iterations, commanded changes in k_1 (resp. k_2) are limited to 0.5 m/s (resp. 0.5 rad/s). For the EV, $k_1 \in [0, 5]$ m/s and $|k_2| \leq 0.5$ rad; commanded changes in k_1 (resp. k_2) are limited to 0.5 m/s (resp. 0.1 rad).

In the simulation demonstrations u_k is a proportional controller for the Segway and a linear MPC controller for the EV. For the hardware demonstrations of both robots, u_k generates control inputs $u_k(t, k) = k \; \forall \; t \in [0, \tau_{\text{plan}}]$; $u_k(t, k) = s(t, k) k \; \forall \; t \in [\tau_{\text{plan}}, \tau_{\text{plan}} + \tau_{\text{brake}}(k)]$; and $u_k(t, k) = 0 \; \forall \; t \in [\tau_{\text{plan}} + \tau_{\text{brake}}(k), T]$. To find tracking error functions, I simulated (3.1) under u_k for each robot over a variety of initial conditions and desired trajectories and fit g as polynomials satisfying (3.16). For each robot (D_{Ti}) is solved as described in Section 3.5, with (v_i, w_i, q_i) as degree 10 polynomials.

The EV also has a trajectory parameterization for left turns it is allowed to use in the intersec-

tion environment described in §5.3.3.2. The left turn model is:

$$f_i(t, z(t), k) = s(t, k) \begin{bmatrix} 1 - \frac{k_2}{l} y(t) \\ \frac{k_2}{l} x(t) \end{bmatrix}, \tag{5.10}$$

$$s(t,k) = \begin{cases} v_0 + at, & t \in \mathbf{t}_1(k) \\ k_1, & t \in \mathbf{t}_2(k) \\ k_1 - a_{\text{brake}}(t - \tau_1(k) - \tau_2(k)), & t \in \mathbf{t}_3(k) \\ 0, & t \in \mathbf{t}_4(k). \end{cases} \tag{5.11}$$

Then, the time intervals $\mathbf{t}_1, \ldots, \mathbf{t}_4 : K \to \mathcal{P}(T)$ are

$$\mathbf{t}_1(k) = [0,\ \tau_1(k)] \tag{5.12}$$

$$\mathbf{t}_2(k) = [\tau_1(k),\ \tau_1(k) + \tau_2(k)], \tag{5.13}$$

$$\mathbf{t}_3(k) = [\tau_1(k) + \tau_2(k),\ \tau_1(k) + \tau_2(k) + \tau_3(k)] \tag{5.14}$$

$$\mathbf{t}_4(k) = [\tau_1(k) + \tau_2(k) + \tau_3(k),\ \tau_4(k)]. \tag{5.15}$$

The times $\tau_1, \ldots, \tau_4 : K \to \mathbb{R}_{\geq 0}$ are:

$$\tau_1(k) = \tfrac{k_1 + v_0}{a}, \quad \tau_2(k) = k_3, \quad \tau_3(k) = \tfrac{k_1}{a_{\text{brake}}}, \quad \tau_4(k) = T, \tag{5.16}$$

so that $[0, T] = \bigcup_{i=1}^{4} \mathbf{t}_i$ by construction. In this implementation $v_0 = 0$ m/s, $a = a_{\text{brake}} 2.0$ m/s^2, and $T = 7.5$ s. Trajectories of (5.10) are arcs of constant curvature, with a speed profile consisting of 4 phases: acceleration, constant speed, braking, and stopped. The parameters $k_1,\ k_2,\ k_3$ are speed, approximate front wheel angle, and duration of the constant speed phase. The parameter ranges are $k_1 \in [4, 5]$ m/s, $k_2 \in [0.15, 0.30]$ rad, and $k_3 \in [1.0, 1.5]$ s. The initial condition range for the FRS is velocities and wheel angles of 0-2 m/s and -0.1 to 0.1 rad. For this (D_{T_s}) is solved with SOS using degree 8 polynomials.

In the planning implementation will implement both the discrete time collision check §4.3.1, referred to as RTD-D, and the interval time collision check §4.3.2, referred to as RTD-I. For RTD-I (D_I) is solved for time intervals of length 0.5 s.

5.3.3 Environments

This section describes the environments used in the Simulation comparisons.

5.3.3.1 Random

For the Segway, simulations in the random environment are in a 20×10 m^2 world with 1–10 0.3×0.3 m^2 box-shaped obstacles. 100 trials were run for each number of obstacles (1000 trials total). In each trial, a random start and goal are chosen approximately 18 m apart. Each obstacle moves at a random constant speed, up to 1 m/s, along a random piecewise-linear path. Simulations are identical for the EV, but the world is 60×10 m^2, and the obstacles are 1×1 m^2 and can travel up to 2 m/s. Conservative predictions are given all planners as defined in Definition 31, and are represented as polygons as in Assumption 43.

5.3.3.2 Intersection

The second scenario requires an unprotected left turn at a 4-way intersection, followed by driving straight for 30 m (see Figure 5.6). 100 random scenarios were generated with lane widths and corner radii of 3.5–4.0 m. At any time, up to 4 obstacle cars (length 2.5–4.0 m and width 1.25–2 m) travel along randomly chosen lanes, and randomly choose to turn, at constant speeds of up to 7 m/s. Up to 2 pedestrians randomly cross one of the four cross walks up to 2 m/s. The ego vehicle starts in the right lane either at the intersection or 30 m away, with initial speed and wheel angle of 0. The cost function for all planners is to minimize the distance to a waypoint placed along the lane centerline. A constraint is added to ensure that the end of any planned trajectory is in a lane (i.e., not in the intersection), so that the vehicle only begins turning if the entire maneuver is validated. Conservative predictions are given all planners as defined in Definition 31, and are represented as polygons as in Assumption 43. For RTD-I predictions satisfying Definition 55 are generated as polygons that bound the obstacles' future motion.

5.3.4 Trajectory Planner Implementations

This section describes implementations of RTD, State lattice [McN11], and linear MPC planning algorithms. RTD will be compared to state-lattice in the random environment and linear MPC in the intersection environment.

5.3.4.1 RTD

For both robots the discrete obstacle representation discussed in §4.3.1 with $b = 0.1$ is used and the quantities r and a derived from Examples 51 and 50. For RTD-D $\tau_{\text{disc}} = 0.1$ s. In the random environment these correspond to buffers of $b_{\text{disc}} = 0.15$ m for the Segway and $b_{\text{disc}} = 0.35$ m for the EV as per Lemma 53. As noted in §5.3.2, RTD-I collison checks over time intervals of 0.5 s. For online planning Algorithm 1 Line 7 is implemented with MATLAB's `fmincon`.

Environment	Platform	Planner	AFC (%)	Goals (%)	Avg Speed (m/s)
Random	Segway	RTD-D	**0.0**	**100.0**	1.18
		State Lattice	7.6	92.4	**1.37**
	EV	RTD-I	**0.0**	**96.8**	**3.06**
		RTD-D	**0.0**	90.7	1.99
		State Lattice	17.2	77.3	2.88
Intersection	EV	RTD-I	**0.0**	**99.0**	2.71
		RTD-D	**0.0**	91.0	1.50
		Linear MPC	19.0	80.0	**3.35**

Table 5.6: Simulation results for RTD in dynamic environments. 1,000 trials are run in the random environment, 100 for the intersection environment. The metrics compared from left to right are At-fault Collisions (AFC), Goals, and Average Speed.

5.3.4.2 State Lattice

A state lattice mid-level planner is implemented as in [McN11] in MATLAB with braking as a fail-safe in each plan and LazySP for searching the lattice graph online [DS16]. Similar to the approach in §5.1.4.1, SL was tested with obstacles buffered by increasing amounts until the planner had collisions in less than 10% (resp. 20%) of trials for the Segway (resp. EV); the final values were 0.43 m (resp. 2.77 m) for the Segway (resp. EV). Since SL planners require feedback about the pose of the generated trajectories, a linear MPC controller is used for both robots.

5.3.4.3 Linear MPC

For the intersection environment I tested a planner where the linear MPC controller tracks lane centerlines joined by a circular arc through the intersection. At each planning iteration, a reference trajectory is generated from the centerlines and arc that is kinematically feasible as in [McN11, Section 3.8], and does not intersect any obstacles. A collision check is performed every 50 ms. I validated that this method can always traverse the intersection when no obstacles are present.

5.3.5 Simulation Results

RTD-I, RTD-D and State Lattice [McN11] planners are compared in the random environment. RTD-I, RTD-D and Linear MPC planners are compared in the intersection environment. In all simulations $t_{\text{plan}} = 0.5$ s; simulations are stopped if the ego vehicle has an at-fault collision or exceeds 300 planning iterations. Results are summarized in Table 5.6.

RTD-D and RTD-I have no at-fault collisions for either robot in either environment. The results of the random environment are discussed first. Collisions occur with state lattice because the robot

cannot perfectly track its reference trajectory, and it is unclear how to buffer obstacles to provably compensate for tracking error and the robot's footprint (a variety of heuristics are presented in [McN11]). Compared to the Segway simulations, the EV simulations are more difficult because the velocity of the robot and obstacles is higher, leading to more collisions for state lattice. Both planners stop more often than in the Segway simulations. Note that the EV is not allowed to reverse and cannot turn in place, so it sometimes gets trapped by obstacles. In the EV simulation RTD-I has more goals reached, and higher average speed than both State Lattice and RTD-D. Importantly RTD-D has the lowest average speed for both platforms. This confirms that RTD-I is able to plan fast but safe trajectories, and highlights the benefits of eliminating the temporal buffer and collision checking over time intervals.

The intersection environment is discussed next. A trial of RTD-I is shown in Figure 5.6. RTD-I completes 99% of the left turns with an average speed of 2.71 m/s; in the one incomplete turn, it was stuck waiting to turn due to aggressive obstacles. RTD-D is able to complete 91% of the scenarios, but with an average speed of 1.5 s, indicating it stays stopped for longer at the intersection. Linear MPC only completes 80% of the left turns, and has at-fault collisions 19% of the time, confirming that the trajectory tracking controller is insufficient on its own. When MPC is successful, it has a higher speed than RTD-I or RTD-D, indicating that, in some scenarios, RTD increases conservatism to prevent collisions. Altogether, these results confirm that RTD-I is capable of validating and executing long maneuvers without causing at-fault collisions.

5.3.6 Hardware Demonstration

To illustrate the capability of RTD-D, was also tested on the Segway (`https://youtu.be/9mMZyyLUiPg`) and EV (`https://youtu.be/PGBxoPMRvg8`) as described above. The Segway runs indoors at up to 1.5 m/s in similar scenarios as in simulation. Virtual dynamic obstacles ($v_{\text{obs,max}} = 1$ m/s) are created in MATLAB. The testing area is smaller than the simulation world, so tests contained up to 3 obstacles. The room boundaries are treated as static obstacles as in §4.2.2. The EV runs outdoors in a large open area at up to 3 m/s, with a safety driver. For the EV, more structured, car-like scenarios were tested by showing a variety of overtake maneuvers. Virtual obstacles ($v_{\text{obs,max}} = 1.5$ m/s) resembling people or cyclists are created in MATLAB. The area is large enough that static obstacles did not need to be considered.

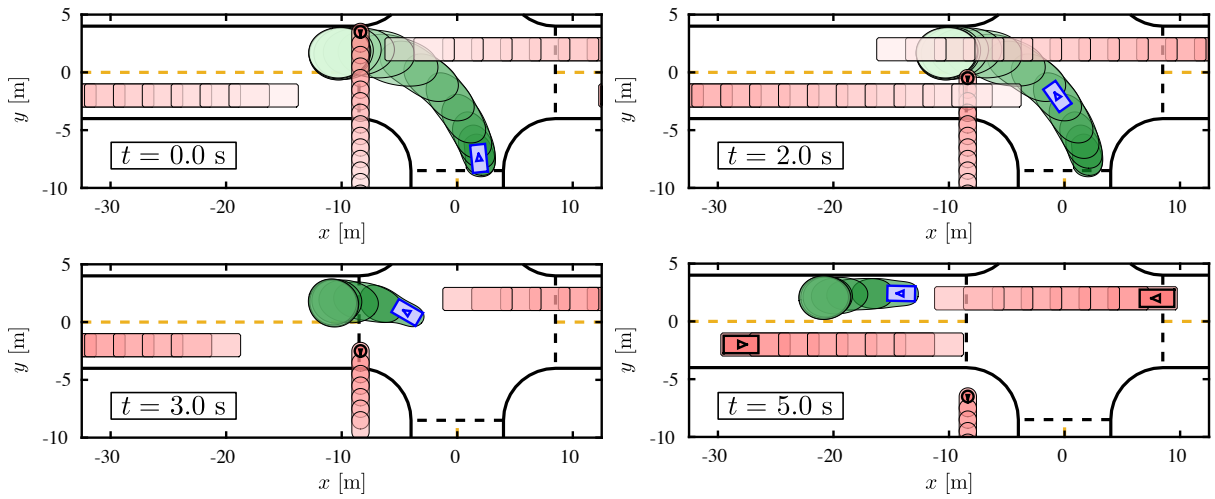


Figure 5.6: Timelapse of EV (blue) completing a left turn. Figures show time at 0.0, 2.0, 3.0, and 5.0 s from top to bottom. Obstacles and their prediction are plotted in red. The vehicle obstacles are traveling at 5 m/s. The pedestrian is traveling at 2 m/s. The ego vehicle begins the scenario stopped at the intersection. The FRS intervals are shown in green. Obstacle predictions and the FRS intervals fade from dark to light with increasing time. The left turn maneuver is longer in duration, and therefore requires longer predictions, than the driving-straight maneuvers (which begin after the ego vehicle completes the turn at $t = 3.0$ s).

CHAPTER 6

Chance-constrained Optimization

This chapter will focus on developing a chance-constrained optimization algorithm, that can let motion planners deal with uncertainty in other obstacles' motion probabilistically. Approaches that make guarantees about safety [VLK+19, PA18] require predictions of obstacles at current and future timesteps that are conservative, meaning that they must contain all possible actions for the obstacles, regardless of how likely they are to occur. In dynamic environments, using obstacle predictions that contain all feasible actions for the obstacle can make finding a feasible solution to trajectory optimizations problems difficult, if not impossible [SSSS17]. Even in the static case, states of obstacles are uncertain due to sensory limitations [Thr02], and one must devote more resources to sensing to increase accuracy and precision.

To limit the probability of an unsafe event, one can use chance-constrained programming where the probability of constraint violation is enforced [Pré13, WHJW20, CLLSA+20]. Tangential approaches that limit the severity of constraint violations using risk measures have also been developed and applied to planning and control problems [DAB20, HY20]. This chapter adopts the chance-constrained approach and proposes the Chance-constrained Parallel Bernstein Algorithm (CCPBA). §6.1 describes the related work. §6.2 proposes the algorithm. §6.3 explains how to apply CCPBA to trajectory optimization problems. §6.4.1 gives simulation comparison to the state-of-the-art. §6.4.2 describes a hardware demonstration of CCPBA working the Segway platform from Figure 3.2a. §6.4.3 concludes with a brief discussion of the pros and cons of CCPBA and possible research directions.

6.1 State-of-the-art

Chance-constrained programming was first introduced by [CCS58] and has been extensively studied ever since [Pré13]. It has been used in applications such as portfolio optimization [KPU02] and for optimal control of chemical processes [HLM+01]. Its application to trajectory optimization has been a more recent development. This section describes state-of-the-art methods for chance-

constrained optimization, with a focus on trajectory optimization. Compared to its early applications, trajectory optimization applications are especially challenging because time limits of 0.5 s or less are often imposed [DWE20] Additionally, their constraints are often nonconvex and can be difficult to evaluate analytically in the chance-constrained sense. Algorithms for chance-constrained programming can be classified into two classes of approaches based upon how they bound constraint violation. The first class of approaches conservatively bounds the probability of constraint violation. The second class of approaches numerically approximate the probability of constraint violation.

6.1.1 Moment-based Bounds

Most methods in the first class of approaches use moment-based inequalities from probability theory which are able to bound the tail of a probability distribution. Cantelli's inequality [Can29], for instance, uses the mean and variance to upper bound the probability of constraint violation. Others have even used the first four central moment to generate an upper bound [Bha87]. Sums-of-squares (SOS) programs can also be solved to compute the upper bound with higher order moments [WHJW20]. In the case where the constraint functions are polynomial, one only needs to compute the moments for the uncertain variables, which can be done efficiently for certain distributions, such as the exponential family. After the moments are computed, the final constraint is a polynomial in the decision variables. In the context of motion planning, this constraint can easily be incorporated in a nonlinear Model Predictive Control programs [WJW20]. Unfortunately, the upper bound of constraint violation that is generated from moment-based approaches can be overly conservative because the bound must hold for any distribution that has the specified moments. Such inequalities can be problematic for use in applications such as autonomous driving where robots operate in narrow corridors.

6.1.2 Numerical Methods

The second class of methods applies numerical integration techniques to bound the probability of constraint violation. A chance-constraint can be equivalently represented as an the expectation of an indicator function on the set of variables that violate the constraint. Monte Carlo approaches, for instance, draw random samples from the uncertainty vector and approximate the integral as a weighted sum [AG07]. Monte Carlo methods are simple to implement; however, their convergence rates can be slow and large numbers samples can lead to long computation times. Additionally the integrand, an indicator function, is not smooth, so Monte Carlo is difficult to use in gradient-based optimization solvers; hence, in the motion planning context they are used in a guess-and-check framework where a planned trajectory is generated, then the probability of constraint violation is

calculated [JSP18].

One can smoothly over-approximate the integrand to apply such Monte-Carlo methods in combination with a gradient-based method to solve such optimization programs [NS07, ZK14]. Such an approach has been utilized in Model Predictive Control (MPC) programs [ZK17, KMKR20]. These methods provide tighter bounds on the probability of constraint violation than the moment-based inequalities; however, they still require numerical evaluation of integrals. For motion planning, one must balance the tightness of the over approximation (looser overapproximations generally result in smoother integrands and faster evaluation) with the desired accuracy and run time constraints.

6.1.3 Special Cases

In cases where the constraints are defined by linear inequalities and the uncertain components are normally distributed, [BOW11] formulates the motion planning problem as a disjunctive convex program. This method relies on using the cumulative distribution function for the normal distribution, which can be evaluated analytically. It is exact in the case of convex constraints. However, for non-convex constraints, such as obstacle avoidance, Boole's inequality is used to upper bound the probability of constraint violation. Drawbacks of this approach are that solving the disjunctive convex program can be slow for motion planning applications and Boole's inequality introduces conservatism around the vertices of non-convex constraint sets.

6.2 Chance-constrained Parallel Bernstein Algorithm

This section presents a novel algorithm for chance-constrained polynomial optimization, called the Chance-constrained Parallel Bernstein Algorithm (CCPBA), which can be applied to a motion planning problems. The algorithm is similar in nature to deterministic optimization programs using Bernstein polynomials [NA11, KZZV20]; however, to the best of my knowledge, this is the first use of Bernstein polynomials for chance-constrained optimization. These algorithms fall under the class of branch-and-bound algorithms, and work by computing lower and upper bounds on the cost and constraint functions for subsets of the decision variables as they search for optimal solutions.

For chance-constraints, the algorithm projects each constraint function onto the Bernstein polynomial basis, which is used to compute upper and lower bounds on the probability of constraint violation by conservatively evaluating the expectation of an indicator function on the constraint set. The advantage of my approach is that it provides certified bounds that are tightened with increased computations. Existing approaches either give a conservative upper bound, that can never converge to the true value; or numerically integrate to approximate the true value, which always

requires large amounts of computations for accuracy. The algorithm can also return suboptimal, feasible solutions during its search process; which is useful for motion planning as it can be implemented as an anytime algorithm [Zil96].

Figure 6.1 shows CCBPA planning a lane change maneuver for an autonomous vehicle whose probability of collision with an adjacent vehicle is guaranteed to be below a user specified threshold. In this example that the probability of collision is small ($< 1\%$), but non-zero. If one were to consider a conservative prediction, that accounted for the minimum and maximum accelerations the other vehicle would apply, such a trajectory would not be feasible.

The rest of the section is organized as follows. §6.2.1 formulates the chance-constrained optimization program. §6.2.2 introduces notation and operations associated with Bernstein polynomials. §6.2.3 describes how Bernstein polynomials are useful for chance-constrained programming. §6.2.4 details the proposed algorithm.

6.2.1 Problem Formulation

This work focuses on solving chance constrained programs of the following form:

$$\begin{aligned} &\min_{k \in K} J(k) \\ &\text{s.t. } \Pr\left(\left\{\xi \in \Xi \;\middle|\; \bigvee_{i \in \{1,\ldots,n_{\text{cons}}\}} (v_i(k,\xi) \geq 0)\right\}\right) \leq \Delta \end{aligned} \tag{6.1}$$

where $k \in K \subset \mathbb{R}^{n_K}$ is a decision variable, $J : \mathbb{R}^{n_K} \to \mathbb{R}$ is the cost function, $(\Xi, \mathcal{F}, \Pr)$ is a probability space, $\xi \in \Xi$ is a possible outcome, $v_i : K \times \Xi \to \mathbb{R}$ are measurable functions. $\vee$ is the logical *or*, meaning the probability that any constraint is violated is limited.

To deal with the joint nature of the chance constraints I used Boole's inequality [NS07, BLW06, OW08], which states that for any finite or countable set of events, $\{A_i\}_{i=1}^m$, with $A_i \in \mathcal{F}$, the probability that at least one event happens is not greater than the sum of the probabilities of the individual events (i.e., $\Pr(\bigcup_{i=1}^m A_i) \leq \sum_{i=1}^m \Pr(A_i)$). Boole's inequality yields a conservative version of (6.1):

$$\begin{aligned} &\min_{k \in K} J(k) \\ &\text{s.t. } \sum_{i=1}^{n_{\text{cons}}} \Pr\left(\{\xi \in \Xi \mid v_i(k,\xi) \geq 0\}\right) \leq \Delta. \end{aligned} \tag{6.2}$$

Note that (6.2) is easier to solve than (6.1) since the probability of violation for each constraint is computed separately. The algorithm will also exploit the fact that for each $k \in K$, the probability of

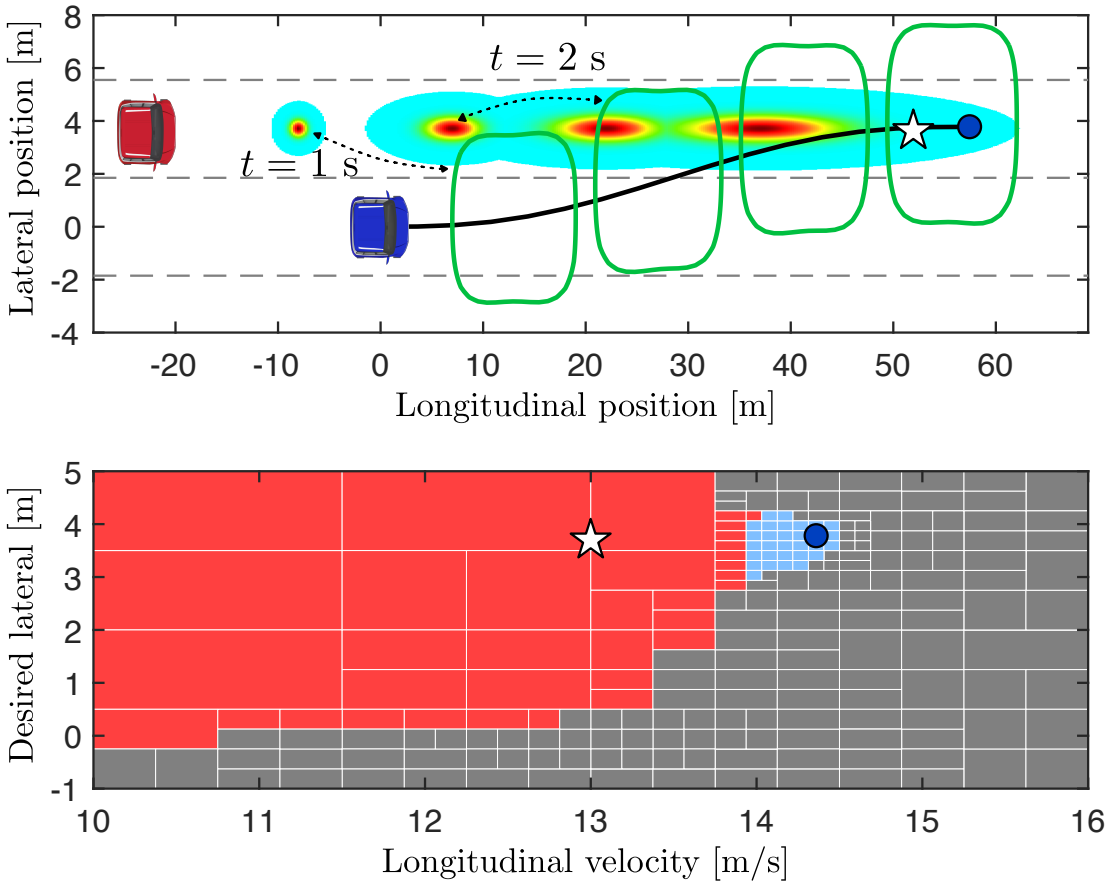


Figure 6.1: Chance-constraint Parallel Bernstein Algorithm (CCPBA) solving a motion planning problem for an autonomous vehicle with a probability limit of 1%. A GPU implementation of CCPBA solves this problem in 0.22 s. The top plot shows the obstacle (red) and probability densities of its center of mass at timesteps of 1,2,3, and 4 s. The confidence ellipsoid containing 5 standard deviations of the densities are colored with red being more likely, and cyan less likely. The ego vehicle (blue) and reachable sets representing collision states (green) for the solution trajectory (black) and endpoint (blue circle) are plotted. The white star is the optimal endpoint. The bottom plot shows the decision variable (longitudinal velocity and desired lateral position) space. Light blue patches represent active patches that could contain a cheaper solution. Red patches have been discarded due to infeasibility. Gray patches have been discarded due to high cost. The blue circle and white star are the current solution and desired parameters.

the i^{th} constraint being violated is equivalent to the expectation of the random variable $\mathbb{1}\big(v_i(k,\xi)\big)$ because this expectation can be written as the following integral:

$$\Pr\big(\{\xi \in \Xi \mid v_i(k,\xi) \geq 0\}\big) = \int_{\Xi} \mathbb{1}\big(v_i(k,\xi)\big) d\Pr. \tag{6.3}$$

6.2.2 Preliminaries

This section presents general notation and operations associated with Bernstein polynomials. x is used as a generic variable in this section. A polynomial, $h : \mathbb{R}^n \to \mathbb{R}$ can be represented in the Bernstein basis over an arbitrary box $\mathbf{x}$ as

$$h(x) = \sum_{I \leq D} B_I^D(\mathbf{x}) b_I^D(\mathbf{x}, x), \tag{6.4}$$

where $b_I^D(\mathbf{x}, \cdot)$ is the I^{th} multivariate Bernstein polynomial of multi-degree D over $\mathbf{x}$, given by [Ham18, (3.15)]

$$b_I^D(\mathbf{x}, x) := \prod_{r=1}^{n} \binom{d_r}{i_r} (x_r - \underline{x}_r)^{i_r} (\overline{x}_r - x_r)^{d_r - i_r} (\overline{x}_r - \underline{x}_r)^{-d_r} \tag{6.5}$$

and $B_I^D(\mathbf{x})$ are the corresponding Bernstein coefficients of h over $\mathbf{x}$ determined by [Ham18, (3.16,3.17)].

All Bernstein coefficients are collected in a multi-dimensional array $B(\mathbf{x}) := (B_I^D(\mathbf{x}))_{I \leq D}$. Let $\min B(\mathbf{x})$ (resp. $\max B(\mathbf{x})$) denote the minimum (resp. maximum) element in the patch $B(\mathbf{x})$. Importantly, the range of h over $\mathbf{x}$ is contained within the interval spanned by the extrema of $B(\mathbf{x})$:

Lemma 56. *[NA11, Lemma 2.2] Let h be a polynomial as defined in (1.1) and $B(\mathbf{x})$ be its bernstein patch over a box $\mathbf{x}$. Then, the following property holds:*

$$\min B(\mathbf{x}) \leq h(x) \leq \max B(\mathbf{x}),\ \forall x \in \mathbf{x}. \tag{6.6}$$

The range enclosure can be improved by subdividing $\mathbf{x}$ into smaller subboxes and computing the Bernstein patches over these subboxes. A *subdivision* in the r^{th} direction ($1 \leq r \leq n$) is a bisection of $\mathbf{x}$ perpendicular to this direction. That is, let

$$\mathbf{x} := [\underline{x}_1, \overline{x}_1] \times \cdots \times [\underline{x}_r, \overline{x}_r] \times \cdots \times [\underline{x}_n, \overline{x}_n] \tag{6.7}$$

be an arbitrary box over which the Bernstein patch $B(\mathbf{x})$ is already computed. Subdividing $\mathbf{x}$ in

the r^{th} direction creates two subboxes $\mathbf{x}_L$ and $\mathbf{x}_R$, defined as

$$\begin{aligned}\mathbf{x}_L &= [\underline{x}_1, \overline{x}_1] \times \cdots \times [\underline{x}_r, (\underline{x}_r + \overline{x}_r)/2] \times \cdots \times [\underline{x}_n, \overline{x}_n], \\ \mathbf{x}_R &= [\underline{x}_1, \overline{x}_1] \times \cdots \times [(\underline{x}_r + \overline{x}_r)/2, \overline{x}_r] \times \cdots \times [\underline{x}_n, \overline{x}_n].\end{aligned} \tag{6.8}$$

Note that I have subdivided $\mathbf{x}$ by halving its width in the r^{th} direction; I choose $1/2$ as the subdivision parameter in this work, but one can choose a different value in $(0, 1)$ (see [NA11, (10)]). The Bernstein patches for the new subboxes $B(\mathbf{x}_L)$ and $B(\mathbf{x}_R)$, can be computed with linear transformations [NA11, §2.2]

$$B(\mathbf{x}_L) = M_{r,L} B(\mathbf{x}), \tag{6.9}$$

$$B(\mathbf{x}_R) = M_{r,R} B(\mathbf{x}), \tag{6.10}$$

where $M_{r,L}$ and $M_{r,R}$ are constant matrices that can be precomputed offline. By repeatedly applying the subdivision procedure and Lemma 56, the bounds on the range of a polynomial in a subbox are improved, and can be shown to be exact in the limiting case:

Theorem 57. *[KZZV20, Theorem 4, Corollary 5] Let $\mathbf{x}$ be a box of maximum width $|\mathbf{x}| = 2^{-l}$ ($l \in \mathbb{N}$) and let $B(\mathbf{x})$ be the corresponding Bernstein patch of a given polynomial h, then*

$$\min B(\mathbf{x}) \leq \min_{x \in \mathbf{x}} h(x) \leq \min B(\mathbf{x}) + \gamma 2^{-2l} \tag{6.11}$$

$$\max B(\mathbf{x}) - \gamma 2^{-2l} \leq \max_{x \in \mathbf{x}} h(x) \leq \max B(\mathbf{x}) \tag{6.12}$$

where γ is a non-negative constant that is independent of l.

In the rest of the paper, multiple functions will be projected onto the Bernstein basis. In subsequent sections the notation $B_h(\mathbf{x})$ can be meant to indicate the Bernstein coefficients from polynomial h on subbox $\mathbf{x}$. There will also be functions of variables from different spaces, for example, decision variables and outcomes from a probability space. In this case if for a polynomial v that is a function of variables k and ξ, its Bernstein coefficients over subboxes $\mathbf{k}$ and $\boldsymbol{\xi}$ will be written as $B_v(\mathbf{k}, \boldsymbol{\xi})$.

6.2.3 Bernstein Polynomials for Chance-constrained Programming

This section describes how Bernstein polynomials are useful for solving (6.2). §6.2.3.1 focuses on chance constraint evaluation, and §6.2.3.2 focuses on evaluation of the objective function.

6.2.3.1 Bernstein Representation of Chance Constraints

This subsection explains how Bernstein Polynomials can be used to compute lower and upper bounds on the probability of constraint violation (6.3). The following assumption about the constraint functions is required:

Assumption 58. *Each* $v_i : K \times \Xi \to \mathbb{R}$ *for* $i \in \{1, \ldots, n_{cons}\}$ *are polynomial functions,*

in addition to the following assumption about the sample space:

Assumption 59. *The sample space* $\Xi \subset \mathbb{R}^m$ *is a* m*-dimensional box.*

These pair of assumptions allows for the constraint functions in (6.2) to be represented in the Bernstein basis set. I next make an assumption about the probability measure:

Assumption 60. *The joint cumulative distribution function for the probability measure* $\Pr$ *exists; hence if* $\boldsymbol{\xi}$ *is a subbox of* Ξ *there is a function,* F*, that returns the probability mass in* $\boldsymbol{\xi}$*:*

$$F(\boldsymbol{\xi}) = \Pr(\xi \in \boldsymbol{\xi}). \tag{6.13}$$

The algorithm will project v_i on the Bernstein basis and F will be used to compute the probability mass in subboxes of Ξ as (6.3) is evaluated. The bounding property presented in Lemma 56 ensures that lower and upper bounds on (6.3) are obtained. Let $\mathbf{k}$ refer to any box in $\mathbb{R}^{n_K}$, Ξ be divided into n m-dimensional, non-overlapping subboxes such that $\Xi = \bigcup_{j=1}^{n} \boldsymbol{\xi}^{(j)}$ and $B_{v_i}(\mathbf{k}, \boldsymbol{\xi}^{(j)})$ refer to the Bernstein patch for the ith constraint and jth subbox of Ξ. The algorithm checks the extrema of each Bernstein patch to determine whether or not they can contribute to the evaluation of (6.3). Define the set of indices corresponding to *decided* patches as

$$\mathcal{D}(\mathbf{k}) = \big\{(i,j) \in \{1, \ldots, n_{\text{cons}}\} \times \{1, \ldots, n\} \mid \min B_{v_i}(\mathbf{k}, \boldsymbol{\xi}^{(j)}) \geq 0\big\}, \tag{6.14}$$

and *undecided* patches as

$$\mathcal{U}(\mathbf{k}) = \big\{(i,j) \in \{1, \ldots, n_{\text{cons}}\} \times \{1, \ldots, n\} \mid \max B_{v_i}(\mathbf{k}, \boldsymbol{\xi}^{(j)}) \geq 0, \min B_{v_i}(\mathbf{k}, \boldsymbol{\xi}^{(j)}) < 0\big\}. \tag{6.15}$$

These sets construct upper and lower bounds on (6.3) as follows:

Theorem 61. *Let* $(\Xi, \mathcal{F}, \Pr)$ *be a probability space. Let* Ξ *be divided into* n m*-dimensional, non-overlapping subboxes such that* $\Xi = \bigcup_{j=1}^{n} \boldsymbol{\xi}^{(j)}$*. Let* $v : K \times \Xi \to \mathbb{R}$ *be a measurable polynomial function. Let* $\mathbf{k}$ *be any box in* K*. Let* $\mathcal{D}$ *and* $\mathcal{U}$ *be defined as in* (6.14) *and* (6.15)*. Then for any*

$k \in \mathbf{k}$ *the following two inequalities hold:*

$$\sum_{(i,j)\in\mathcal{D}(\mathbf{k})} F(\boldsymbol{\xi}^{(j)}) \leq \sum_{i=1}^{n_{\text{cons}}} \Pr\left(\{\xi \in \Xi \mid v_i(k,\xi) \geq 0\}\right) \leq \sum_{(i,j)\in\mathcal{D}(\mathbf{k})\cup\mathcal{U}(\mathbf{k})} F(\boldsymbol{\xi}^{(j)}). \tag{6.16}$$

Proof. First note that if Ξ is partitioned into n subboxes, the integral of any function can be computed by summing the integral over each subbox

$$\sum_{i=1}^{n_{\text{cons}}} \int_{\Xi} \mathbb{1}\big(v_i(k,\boldsymbol{\xi})\big) d\Pr = \sum_{i=1}^{n_{\text{cons}}} \sum_{j=1}^{n} \int_{\boldsymbol{\xi}^{(j)}} \mathbb{1}\big(v_i(k,\boldsymbol{\xi})\big) d\Pr\,. \tag{6.17}$$

To prove the upper bound observe that for any $(i,j) \in \{1,...,n_{\text{cons}}\} \times \{1,...,n\}$

$$\int_{\boldsymbol{\xi}^{(j)}} \mathbb{1}\big(v_i(k,\xi)\big) d\Pr \leq \mathbb{1}\left(\max_{(k,\xi)\in\mathbf{k}\times\boldsymbol{\xi}^{(j)}} v_i(k,\xi)\right) F(\boldsymbol{\xi}^{(j)}), \tag{6.18}$$

and then

$$\mathbb{1}\left(\max_{(k,\xi)\in\mathbf{k}\times\boldsymbol{\xi}^{(j)}} v_i(k,\xi)\right) F(\boldsymbol{\xi}^{(j)}) \leq \mathbb{1}(\max B_{v_i}(\mathbf{k},\boldsymbol{\xi}^{(j)}))F(\boldsymbol{\xi}^{(j)}) \tag{6.19}$$

The last inequality follows from Lemma 56 which shows that $\{\max v_i(k,\xi) : (k,\xi) \in \mathbf{k}\times\boldsymbol{\xi}^{(j)}\} \leq \max B_{v_i}(\mathbf{k},\boldsymbol{\xi}^{(j)})$.

I have proven the upper bound is true for the ith constraint and j^{th} subbox, and it is known by (6.14,6.15) that $\mathcal{D}(\mathbf{k}) \cup \mathcal{U}(\mathbf{k})$ consists of patches where $\mathbb{1}\big(\max B_{v_i}(\mathbf{k},\boldsymbol{\xi}^{(j)})\big) > 0$. Therefore one can deduce that the upper bound holds as the sum over all constraints and subboxes is taken. The lower bound can be found by replacing the max with min, switching the order of the inequalities, and only considering decided patches in $\mathcal{D}$. □

The algorithm will iteratively subdivide Ξ, and rely on the fact that the lower and upper bounds on (6.3) become tighter as the Bernstein approximation of the constraint functions become more accurate via Theorem 57. This section concludes with a remark about the sample space.

Remark 62. *Assumption 59 means that Ξ has finite width, which may not hold for some types of probability distributions, such as those from the exponential family. The algorithm can still be applied; however one must truncate the sample space to a box of finite width, then reduce the threshold Δ by $n_{cons}(1 - F(\Xi))$, to account for probability mass that lies outside of the truncated domain.*

6.2.3.2 Bernstein Representation of the Objective Function

Bernstein polynomials will also prove useful to represent the objective function and solve (6.2) to global optimality. The following assumptions are required:

Assumption 63. *Each $J : K \to \mathbb{R}$ is a polynomial function;*

Assumption 64. *The decision variable space $K \subset \mathbb{R}^{n_K}$ is a n_K-dimensional box.*

By projecting J onto the Bernstein basis, the maximum and minimum Bernstein coefficients are used to find upper and lower bounds of the objective function in a subbox; subdivision can be performed to get tighter bounds; allowing one to asses the optimality of a subbox. For a thorough analysis of how Bernstein polynomials are useful for solving deterministic polynomial optimization programs to global minima and their convergence properties I refer the reader to [KZZV20].

6.2.4 Algorithm

This section proposes the Chance-Constrained Parallel Bernstein Algorithm (CCPBA, Algorithm 4) to solve (6.2). I extend the approach in [KZZV20] to solve chance-constrained programs. The approach utilizes the Bernstein representation to obtain upper and lower bounds of both objective polynomials (Lemma 56) and chance constraints (Theorem 61), iteratively improves such bounds using subdivision (Theorem 57), and removes patches that cannot contain a solution (Theorem 72). The algorithm, the list used to store patches, tolerances and stopping criteria, subdivision, a cut-off test for eliminating patches, and the advantages and disadvantages of CCPBA are discussed.

6.2.4.1 Items and Lists

The algorithm maintains two lists. The first is for the objective function, which is only a function of the decision variables. *Items* in this list are tuples $(\mathbf{k}, B_J(\mathbf{k}), \Delta_{\text{lo}})$ where $B_J(\mathbf{k})$ is a Bernstein patch associated with the cost function for subbox $\mathbf{k} \subseteq K$ and $\Delta_{\text{lo}} \in \mathbb{R}_{\geq 0}$ is a lower bound on the probability of constraint violation for any $k \in \mathbf{k}$. This *list* $\mathcal{K} = \{\kappa^{(j)} : j = 1, \ldots, n_{\mathcal{K}}\}$, $n_{\mathcal{K}} \in \mathbb{N}$, is indexed by $j \in \mathbb{N}$. Note that elements in items in a list will be referred to, for example, $\mathbf{k} \in \mathcal{K}$ would mean all of the subboxes in items in $\mathcal{K}$. The other list will pertain to the chance constraints, $\mathcal{L}$, where an item is a tuple $(\mathbf{k}, \boldsymbol{\xi}, B_{v_i}(\mathbf{k}, \boldsymbol{\xi}))$ and $B_{v_i}(\mathbf{k}, \boldsymbol{\xi})$ are Bernstein patches associated with constraint functions subbboxes $\mathbf{k} \times \boldsymbol{\xi} \subseteq K \times \Xi$. An item in this list $\mathcal{L} = \{\ell^{(j)} : j = 1, \ldots, n_{\mathcal{L}}\}$, $n_{\mathcal{L}} \in \mathbb{N}$, is indexed by $j \in \mathbb{N}$. To initialize the lists first make the following assumption:

Assumption 65. *Without loss of generality, the domain of the decision variable and sample space are unit boxes (i.e., $K = [0,1]^{n_K}$ and $\Xi = [0,1]^m$); since any nonempty box in $\mathbb{R}^n$ can be mapped affinely onto $[0,1]^n$ [TG17], and K and Ξ are boxes by Assumptions 59 and 64.*

6.2.4.2 Algorithm Summary

Algorithm 4 begins by projecting the cost function and each constraint function onto the Bernstein basis in $K \times \Xi$ and initializing the lists, and counters (Lines 1-3). An iteration begins when available memory is checked (Line 4). Items in $\mathcal{L}$ and $\mathcal{K}$ (if applicable) are subdivided along the r^{th} dimension using (Line 5 and Algorithms 5 and 6). Then bounds on the objective function for the items in $\mathcal{K}$ (Line 6 are found with Algorithm 7). Line 6 (Algorithm 8) is used to find lower and upper and bounds on the constraint functions and compute the probability mass in each subbox $\boldsymbol{\xi}$, using the joint cumulative distribution function in Assumption 60. Then lower (resp. upper) bounds on the chance constraint for each item in $\mathcal{K}$ are found by summing up the probability mass for decided (6.14) (resp. decided and undecided (6.15)) patches (see Line 7, Algorithm 9, and Theorem 61).

In Line 8, Algorithm 10 determines a best estimate of the objective function, and generates lists of items in $\mathcal{K}$ that are, infeasible (Definition 68), suboptimal (Definition 71), or unknown. If every patch is infeasible (Line 9), CCPBA returns that the problem is infeasible (Line 15); otherwise, CCPBA checks if the current solution estimate meets user-specified tolerances (Line 10). If the tolerances are met, CCPBA returns the solution estimate (Line 14). Otherwise, CCPBA determines in $\mathcal{L}$ that can be eliminated (Line 11 and Algorithm 11). CCPBA then eliminates all infeasible and suboptimal items in $\mathcal{K}$ and $\mathcal{L}$ (Line 12 and Algorithm 12), then moves to the next iteration (Line 13). Note, algorithms 5, 6, 7, 8, and 12 can be parallelized. CCPBA is implemented on a GPU in §6.4.

6.2.4.3 Tolerances and Stopping Criteria

Recall that, by Lemma 56, Bernstein patches provide upper and lower bounds for polynomials over a box. From Theorem 57, as $[0,1]^{n_K}$ and $[0,1]^m$ are subdivided into smaller subboxes, the bounds of the Bernstein patches on each subbox more closely approximate the actual bounds of each polynomial. However, to ensure the algorithm terminates, tolerances on optimality must be set. During optimization one also typically wants to find the optimal up to some resolution. This resolution corresponds to the maximum allowable subbox width which is referred to as the *step tolerance*.

Definition 66. *Denote the* optimality tolerance *as* $\epsilon > 0$ *and the* step tolerance *as* $\delta > 0$*. Algorithm 4 is terminated either when* $\mathcal{K}$ *is empty (the problem is infeasible) or when there exists an item* $(\mathbf{k}, B_J(\mathbf{k}), \Delta_{lo}) \in \mathcal{K}$ *that satisfies all of the following conditions:*

(a) $|\mathbf{k}| \leq \delta$*,*

(b) $\max B_J(\mathbf{k}) - \min B_J(\mathbf{y}) \leq \epsilon, \ \forall\, \mathbf{y} \in \mathcal{K}$*,*

Algorithm 4 Chance-constrained Parallel Bernstein Algorithm

Inputs: Cost function J; $\{v_i\}_{i=1}^{n_{\text{cons}}}$; F cumulative density function for ξ, chance constraint limit Δ; optimality tolerance $\epsilon > 0$; step tolerance $\delta > 0$, maximum number of patches $n_{\max} \in \mathbb{N}$ and of iterations $n_{\text{iter}} \in \mathbb{N}$.

Outputs: Estimate cost J^* of optimal solution and subbox $\mathbf{k}^*$

Algorithm:

1: Initialize patches of J and v over n_K and m dimensional unit boxes as in [TG17, (14)]
$[B_J([0,1]^{n_K}), B_{v_i}([0,1]^{n_K},[0,1]^m)] \leftarrow \text{InitPatches}(J, v_i)$

2: Initialize lists of undecided patches and patch extrema on the GPU
$\mathcal{K} \leftarrow ([0,1]^{n_K}, B_J([0,1]^{n_K}), 0), \quad \mathcal{K}_{\text{bounds}} \leftarrow \{\},$
$\mathcal{L} \leftarrow ([0,1]^{n_K}, [0,1]^m, B_{v_i}([0,1]^{n_K},[0,1]^m)), \quad \mathcal{L}_{\text{bounds}} \leftarrow \{\}$

3: Initialize iteration count and subdivision direction
$r \leftarrow 1,\ \text{iter} \leftarrow 1$

4: Test for sufficient memory (iteration begins here)
if $2\text{length}(\mathcal{L}) > n_{\max}$ **then** go to 14 **end if**

5: (Parallel) Subdivide each patch in $\mathcal{K}$ and $\mathcal{L}$ along r^{th} dimension using Algorithms 5 and 6
if $r \leq n$ **then** $\mathcal{K} \leftarrow \text{SubdivideK}(\mathcal{K}, r)$ **endif**
$\mathcal{L} \leftarrow \text{SubdivideL}(\mathcal{L}, r)$

6: (Parallel) Find bounds for J, v_i and density weight in each subbox in $\mathcal{L}$ using Algorithms 7 and 8
if $r \leq n$ **then** $\mathcal{K}_{\text{bounds}} \leftarrow \text{FindBoundsK}(\mathcal{K}, r)$ **endif**
$\mathcal{L}_{\text{bound}} \leftarrow \text{FindBoundsL}(\mathcal{L}, F)$

7: Find chance constraint bounds, and store lower bound in $\mathcal{K}$, using Algorithm 9
$[\mathcal{K}, \Delta_{\text{bounds}}] \leftarrow \text{ChanceConsBounds}(\mathcal{K}, \mathcal{L}_{\text{bounds}})$

8: Estimate upper bound J^*_{up} of the global optimum as the least upper bound of all feasible subboxes and determine which patches of $\mathcal{K}$ to eliminate using Algorithm 10
$[J^*_{\text{lo}}, J^*_{\text{up}}, \mathcal{K}_{\text{save}}, \mathcal{K}_{\text{elim}}] \leftarrow \text{CutOff}(\mathcal{K}_{\text{bounds}}, \Delta_{\text{bounds}})$

9: Test is problem is feasible
if length($\mathcal{K}_{\text{save}}$) = 0 **then** go to 15

10: Test stopping criteria for all $(\mathbf{k}, B_J(\mathbf{k}), \Delta_{\text{lo}}) \in \mathcal{K}$
if $J^*_{\text{up}} - J^*_{\text{lo}} \leq \epsilon$ **and** $|\mathbf{k}| \leq \delta$ **then** go to 14 **end if**

11: Determine which patches of $\mathcal{L}$ to eliminate using Algorithm 11
$[\mathcal{L}_{\text{save}}, \mathcal{L}_{\text{elim}}] \leftarrow \text{CleanL}(\mathcal{L}_{\text{bounds}}, \mathcal{K}, \mathcal{K}_{\text{elim}})$

12: (Parallel) Eliminate infeasible, suboptimal patches using Algorithm 12
$\mathcal{K} \leftarrow \text{Eliminate}(\mathcal{K}, \mathcal{K}_{\text{save}}, \mathcal{K}_{\text{elim}}), \quad \mathcal{L} \leftarrow \text{Eliminate}(\mathcal{L}, \mathcal{L}_{\text{save}}, \mathcal{L}_{\text{elim}})$

13: Prepare for next iteration
$r \leftarrow (\text{mod}(r+1, n+m)) + 1$
if $r = 1$ **then** iter $\leftarrow$ iter $+ 1$ **end if**
if iter $= n_{\text{iter}}$ **then** go to Step 14 **else** go to Step 4 **end if**

14: Return current best approximate solution
$J^* \leftarrow J^*_{\text{up}}, \quad \mathbf{k}^* \leftarrow \mathbf{k}$ for which $\max B_J(\mathbf{k}) = J^*_{\text{up}}$
return J^*, $\mathbf{k}^*$

15: No solution found (problem infeasible)

(c) $\Delta_{\text{lo}} + \sum_{\boldsymbol{\xi} \in \mathcal{L}_{\mathbf{k}}^{\mathcal{U}}} F(\boldsymbol{\xi}) \leq \Delta$*, where*

$$\mathcal{L}_{\mathbf{k}}^{\mathcal{U}} = \{\ell \in \mathcal{L} \mid \ell_1 = \mathbf{k},\ \max(B_{v_i}(\mathbf{k}, \boldsymbol{\xi})) \geq 0, \min(B_{v_i}(\mathbf{k}, \boldsymbol{\xi})) < 0\} \tag{6.20}$$

Feasibility is discussed in more detail in §6.2.4.5. Note that, the step tolerance δ can be implemented by counting the number of iterations, since subdivision halves the width of each subbox.

6.2.4.4 Subdivision

Subdivision for the $\mathcal{K}$ and $\mathcal{L}$ lists is implemented with Algorithms 5 and 6. These algorithms are based off of [KZZV20, Algorithm 2] Since the subdivision of one Bernstein patch is computationally independent of another, in the GPU implementation, each subdivision task is assigned to an individual thread, making Algorithms 5 and 6 parallel.

The subdivision of Bernstein patches can be done in any direction, leading to the question of how to select the direction in practice. For the decision variable space example rules are available in the literature, such as maximum width [RC95, §3], derivative-based [ZG98, §3], or a combination of the two [RC95, §3]. In the context of constrained optimization, the maximum width rule is usually favored [KZZV20].

For the sample space, Ξ, rules for subdivision directions are less clear. In this work, subdivision occurs in the order $1, 2, \ldots, n_K+m$ and the width of each subbox is halved, leading to the following remark.

Remark 67. *In the i^{th} iteration, the maximum width of any subbox in $\mathcal{K}$ and $\mathcal{L}$ is 2^{-i}.*

Exploring rules to help determine subdivision direction and width could be a focus of future work.

Algorithm 5 $\mathcal{K} = \text{SubdivideK}(\mathcal{K}, r)$ (Parallel)

1: $n \leftarrow \text{length}(\mathcal{K})$
2: **parfor** $i \in \{1, \ldots, n\}$ **do**
3: $(\mathbf{k}, B_J(\mathbf{k}), \Delta_{\text{lo}}) \leftarrow \mathcal{K}[i]$
4: Subdivide $\mathbf{k}$ along the r^{th} direction into $\mathbf{k}_L$ and $\mathbf{k}_R$
5: Compute patches $B_J(\mathbf{k}_L)$ and $B_J(\mathbf{k}_R)$
6: $\mathcal{K}[i] \leftarrow (\mathbf{k}_L, B_J(\mathbf{k}_L), \Delta_{\text{lo}})$
7: $\mathcal{K}[i+n] \leftarrow (\mathbf{k}_R, B_J(\mathbf{k}_R), \Delta_{\text{lo}})$
8: **end parfor**
9: **return** $\mathcal{K}$

Algorithm 6 $\mathcal{L} = \text{SubdivideL}(\mathcal{L}, r)$ (Parallel)

1: $n \leftarrow \text{length}(\mathcal{L})$
2: **parfor** $j \in \{1, \ldots, n\}$ **do**
3: $(\mathbf{k}, \boldsymbol{\xi}, B_{v_i}(\mathbf{k}, \boldsymbol{\xi})) \leftarrow \mathcal{L}[j]$;
4: Subdivide $\mathbf{k}$ along the r^{th} direction into $\mathbf{k}_L$ and $\mathbf{k}_R$
5: Subdivide $\boldsymbol{\xi}$ along the r^{th} direction into $\boldsymbol{\xi}_L$ and $\boldsymbol{\xi}_R$
6: Compute patches $B_{v_i}(\mathbf{k}_L, \boldsymbol{\xi}_L)$ and $B_{v_i}(\mathbf{k}_R, \boldsymbol{\xi}_R)$
7: $\mathcal{L}[j] \leftarrow (\mathbf{k}_L, \boldsymbol{\xi}_L, B_{v_i}(\mathbf{k}_L, \boldsymbol{\xi}_L))$
8: $\mathcal{L}[j+n] \leftarrow (\mathbf{k}_R, \boldsymbol{\xi}_R, B_{vi}(\mathbf{k}_R, \boldsymbol{\xi}_R))$
9: **end parfor**
10: **return** $\mathcal{L}$

6.2.4.5 Cut-Off Test

Subdivision leads to an exponential increase in memory useage, however a cut-off test can eliminate items in $\mathcal{K}$ that cannot contain the global minimizer. The potential of items in $\mathcal{K}$ to contain a global minimizer of (6.2) can be determined using the following definitions:

Definition 68. *An item* $(\mathbf{k}, B_J(\mathbf{k}), \Delta_{\text{lo}}) \in \mathcal{K}$ *is* feasible *if*

$$\Delta_{\text{lo}} + \sum_{\boldsymbol{\xi} \in \mathcal{L}_{\mathbf{k}}^{\mathcal{U}}} F(\boldsymbol{\xi}) \leq \Delta \tag{6.21}$$

where $\mathcal{L}_{\mathbf{k}}^{\mathcal{U}}$ *is defined as in* (6.20)*. An item is* infeasible *if*

$$\Delta_{\text{lo}} > \Delta. \tag{6.22}$$

An item is unknown *if it is neither feasible nor infeasible.*

The criteria for feasibility (6.21) and (6.22) is related to the chance constrained program as follows.

Theorem 69. *Let* $(\mathbf{k}, B_J(\mathbf{k}), \Delta_{\text{lo}}) \in \mathcal{K}$ *be an item, and* $\mathcal{L}$ *be a list of patches in* $K \times \Xi$*. If the item is feasible according to Definition 68, then the chance constraint in* (6.2) *is satisfied. If the item is infeasible according to Definition 68, then the chance constraint is not satisfied.*

Proof. In Algorithm 9 Line 13, patches belonging to the decided set (6.14) are added to Δ_{lo} at each iteration, hence Δ_{lo} is equivalent to the lower bound in (6.16) by construction. In Algorithm 9, Lines 14 and 16, patches belonging to the undecided set (6.15) are added to Δ_{lo} at each iteration, hence (6.21) is equivalent to the upper bound in (6.16) by construction. One can then apply Theorem 61 to obtain the desired result. □

The optimality of an item is now discussed.

Definition 70. *The* solution estimate J^*_{up} *is the smallest upper bound of the cost over all feasible items in* $\mathcal{K}$*:*

$$J^*_{\text{up}} = \min \Big\{ \max\{\kappa_2 \mid \kappa \in \mathcal{K},\ \kappa \text{ feasible}\} \Big\}, \tag{6.23}$$

where $\kappa_2 = B_J(\mathbf{k})$ *if* $\kappa = (\mathbf{k},\, B_J(\mathbf{k}),\, \Delta_{\text{lo}})$ *and feasible is defined as in Definition 68.*

Definition 71. *An item* $(\mathbf{k}, B_J(\mathbf{k}), \Delta_{\text{lo}}) \in \mathcal{K}$ *is suboptimal if*

$$\min B_J(\mathbf{k}) > J^*_{\text{up}} \tag{6.24}$$

Any item that is infeasible or suboptimal can be eliminated from $\mathcal{K}$, because the corresponding subboxes cannot contain the solution to the chance constrained program (formalized in the following Theorem). Checking for infeasible and suboptimal items is called *cut-off test*.

Theorem 72. *[KZZV20, Theorem 12] Let* $(\mathbf{k}, B_J(\mathbf{k}), \Delta_{\text{lo}}) \in \mathcal{K}$ *be an item. If the item is infeasible (as in Definition 68) or suboptimal (as in Definition 71), then* $\mathbf{k}$ *does not contain a global minimizer of* (6.2)*. Such item can be removed from the list* $\mathcal{K}$*.*

The cut-off tests is implemented as follows. Algorithms 7 and 8 (FindBounds) computes the maximum and minimum element of each Bernstein patch for the objective function, constraint functions, and the probability mass in each subbox $\boldsymbol{\xi}$. Algorithm 9 determines upper and lower bounds on the chance constraint, which define feasibility via Definition 68 and Theorem 69. Algorithm 10 (CutOffTest) implements the cut-off tests and marks all subboxes to be eliminated with a list $\mathcal{K}_{\text{elim}}$.

In addition to the items in $\mathcal{K}$, items $\mathcal{L}$ that are no longer useful can be eliminated. The following corollary that extends Theorem 72 to items in $\mathcal{L}$:

Corollary 73. *Let* $\ell \in \mathcal{L}$ *and* $\kappa \in \mathcal{K}$ *be items with matching* $\mathbf{k}$ *components. If if* κ *can be removed from* $\mathcal{K}$ *following Theorem 72, then* ℓ *can be removed from* $\mathcal{L}$*.*

Proof. Theorem 72 provides that $\mathbf{k}$ does not contain the global minimizer. □

To further save memory, items in $\mathcal{L}$ that will not further contribute to evaluation of the chance constraints can be removed.

Theorem 74. *Let* $(\mathbf{k}, \boldsymbol{\xi}, B_{v_i}(\mathbf{k}, \boldsymbol{\xi})) \in \mathcal{L}$ *be an item. The item will not affect the evaluation of the chance constraints in* (6.2) *and can be eliminated if it meets any of the following criteria:*

(a) $F(\boldsymbol{\xi}) = 0$

(b) $\max B_{v_i}(\mathbf{k}, \boldsymbol{\xi}) < 0$*;*

(c) $\min B_{v_i}(\mathbf{k}, \boldsymbol{\xi}) \geq 0$*, and Algorithm 4 Line 7 has been executed.*

Proof. Items satisfying criteria (a) can be eliminated because their contribution to the evaluation of the chance constraints in (6.16) and (6.16) is 0, meaning they will have no impact on evaluating the feasibility criteria in Definition 68 (6.21,6.22).

Items satisfying criteria (b) can be eliminated because their constraint functions are definitely negative by Lemma 56; hence they will not correspond to indices in the decided (6.14) nor (6.15) sets; meaning they will not be used to determine feasibility in Definition 68 (6.21,6.22).

Items satisfying criteria (c) have patches that correspond to indices in the (6.14) set. However, in Algorithm 4 Line 7, their contribution to the feasibility criteria (6.21) and (6.22) are stored as Δ_{lo} in items in $\mathcal{K}$ (see Algorithm 9 Line 18), hence they can be removed. For mathematical intuition, recall that the integrand in the evaluation of the chance constraint (6.3) contains an indicator function, so once a patch is identified as non-negative, further subdivision is not necessary. □

Algorithm 11 identifies items in $\mathcal{L}$ that meet these criteria and builds a list $\mathcal{L}_{\text{elim}}$ of those that can be eliminated. Finally Algorithm 12 (Eliminate [KZZV20, Algorithm 5]) eliminates the marked subboxes from the lists $\mathcal{K}$ and $\mathcal{L}$. Algorithms 7, 8 and 12 are parallelizable, whereas Algorithm 10 must be computed serially. Algorithm 11 is done in serial in the presented implementation.

Algorithm 7 $\mathcal{K}_{\text{bounds}} = \text{FindBoundsK}(\mathcal{K})$ (Parallel)

1: $n \leftarrow \text{length}(\mathcal{K})$
2: **parfor** $i \in \{1, \ldots, n\}$ **do**
3: $\quad (\mathbf{k}, B_J(\mathbf{k}), \Delta_{\text{lo}}) \leftarrow \mathcal{K}[i]$
4: $\quad$ Find $\min B_J(\mathbf{k})$ and $\max B_J(\mathbf{k})$ by parallel reduction
5: $\quad \mathcal{K}_{\text{bounds}}[i] \leftarrow (\mathbf{k}, \min B_J(\mathbf{k}),\ \max B_J(\mathbf{k}))$
6: **end parfor**
7: **return** $\mathcal{K}_{\text{bounds}}$

Algorithm 8 $\mathcal{L}_{\text{bounds}} = \text{FindBoundsL}(\mathcal{L}, F)$ (Parallel)

1: $n \leftarrow \text{length}(\mathcal{L})$
2: **parfor** $j \in \{1, \ldots, n\}$ **do**
3: $\quad (\mathbf{k}, \boldsymbol{\xi}, B_{v_i}(\mathbf{k}, \boldsymbol{\xi})) \leftarrow \mathcal{L}[j]$
4: $\quad$ Find $\min B_{v_i}(\mathbf{k}, \boldsymbol{\xi})$ and $\max B_{v_i}(\mathbf{k}, \boldsymbol{\xi})$
5: $\quad \mathcal{L}_{\text{bounds}}[j] \leftarrow (\mathbf{k}, \min B_{v_i}(\mathbf{k}, \boldsymbol{\xi}), \max B_{v_i}(\mathbf{k}, \boldsymbol{\xi}), F(\boldsymbol{\xi}))$
6: **end parfor**
7: **return** $\mathcal{L}_{\text{bounds}}$

This section concludes with following corollary about CCPBA and its ability to find the global minimizer.

Algorithm 9 $[\mathcal{K}, \Delta_{\text{bounds}}] = \text{ChanceConsBounds}(\mathcal{K}, \mathcal{L}_{\text{bounds}})$

$n \leftarrow \text{length}(\mathcal{K})$
for $i \in \{1, \ldots, n\}$ **do**
  $(\mathbf{k}, B_J(\mathbf{k}), \Delta_{\text{lo}}) \leftarrow \mathcal{K}[i]$
  $\Delta_{\text{bounds}}[i] \leftarrow (\Delta_{\text{lo}}, \Delta_{\text{lo}})$
end for
$n \leftarrow \text{length}(\mathcal{L}_{\text{bounds}})$
for $j \in \{1, \ldots, n\}$ **do**
  $(\mathbf{k}, \min B_{v_i}(\mathbf{k}, \boldsymbol{\xi}), \max B_{v_i}(\mathbf{k}, \boldsymbol{\xi}), F(\boldsymbol{\xi})) \leftarrow \mathcal{L}_{\text{bounds}}[j]$
  $l \leftarrow$ index of $\mathcal{K}$ with matching $\mathbf{k}$
  $(\Delta_{\text{lo}}, \Delta_{\text{up}}) \leftarrow \Delta_{\text{bounds}}[l]$
  $(\mathbf{k}, B_J(\mathbf{k}), \Delta_{\text{lo}}) \leftarrow \mathcal{K}[l]$
  if $\min B_{v_i}(\mathbf{k}, \boldsymbol{\xi}) \geq 0$ **then**
    $\Delta_{\text{lo}} += F(\boldsymbol{\xi})$
    $\Delta_{\text{up}} += F(\boldsymbol{\xi})$
  else if $\max B_{v_i}(\mathbf{k}, \mathbf{k}) \geq 0$ **then**
    $\Delta_{\text{up}} += F(\boldsymbol{\xi})$
  end if
  $\mathcal{K}[l] \leftarrow (\mathbf{k}, B_J(\mathbf{k}), \Delta_{\text{lo}})$
  $\Delta_{\text{bounds}}[l] \leftarrow (\Delta_{\text{lo}}, \Delta_{\text{up}})$
end for
return $\mathcal{K}, \Delta_{\text{bounds}}$

Algorithm 10 $[J^*_{\text{lo}}, J^*_{\text{up}}, \mathcal{K}_{\text{save}}, \mathcal{K}_{\text{elim}}] = \text{CutOff}(\mathcal{K}_{\text{bounds}}, \Delta_{\text{bounds}})$

$J^*_{\text{lo}} \leftarrow \infty,\ J^*_{\text{up}} \leftarrow \infty$
$n \leftarrow \text{length}(\mathcal{K}_{\text{bounds}})$
for $i \in \{1, \ldots, n\}$ **do**
  $(\Delta_{\text{lo}}, \Delta_{\text{up}}) \leftarrow \Delta_{\text{bounds}}[i]$
  $(\mathbf{k}, \min B_J(\mathbf{k}), \max B_J(\mathbf{k})) \leftarrow \mathcal{K}_{\text{bounds}}[i]$
  if $\Delta_{\text{up}} \leq \Delta$ **then** $J^*_{\text{up}} \leftarrow \min(J^*_{\text{up}}, \max B_J(\mathbf{k}))$ **endif**
  if $\Delta_{\text{lo}} \leq \Delta$ **then** $J^*_{\text{lo}} \leftarrow \min(J^*_{\text{lo}}, \min B_J(\mathbf{k}))$ **endif**
end for
$\mathcal{K}_{\text{save}} \leftarrow \{\},\ \mathcal{K}_{\text{eliminate}} \leftarrow \{\}$
for $i \in \{1, \ldots, n\}$ **do**
  $(\Delta_{\text{lo}}, \Delta_{\text{up}}) \leftarrow \Delta_{\text{bounds}}[i]$
  $(\mathbf{k}, \min B_J(\mathbf{k}), \max B_J(\mathbf{k})) \leftarrow \mathcal{K}_{\text{bounds}}[i]$
  if $(\Delta_{\text{lo}} \leq \Delta) \wedge (\min B_J(\mathbf{k}) < J^*_{\text{up}})$ **then**
    Append i to $\mathcal{K}_{\text{save}}$
  else
    Append i to $\mathcal{K}_{\text{elim}}$
  end if
end for
return $J^*_{\text{lo}}, J^*_{\text{up}}, \mathcal{K}_{\text{save}}, \mathcal{K}_{\text{elim}}$

Algorithm 11 $[\mathcal{L}_{\text{save}}, \mathcal{L}_{\text{elim}}]$ =CleanL($\mathcal{L}_{\text{bounds}}$,$\mathcal{K}$,$\mathcal{K}_{\text{elim}}$)

Initialize lists for indices of patches to save or eliminate
    $\mathcal{L}_{\text{save}} \leftarrow \{\}$, $\mathcal{L}_{\text{elim}} \leftarrow \{\}$
$n \leftarrow \text{length}(\mathcal{L}_{\text{bounds}})$
for $j \in \{1, \ldots, n\}$ **do**
    $(\mathbf{k}, \min B_{v_i}(\mathbf{k}, \boldsymbol{\xi}), \max B_{v_i}(\mathbf{k}, \boldsymbol{\xi}), F(\boldsymbol{\xi})) \leftarrow \mathcal{L}_{\text{bounds}}[j]$
    if $(\exists\, \kappa \in \mathcal{K}(\mathcal{K}_{\text{elim}})$ with $\kappa_1 = \mathbf{k}) \vee (F(\boldsymbol{\xi}) = 0)$ **then**
        Append k to $\mathcal{L}_{\text{elim}}$
    else
        if $(\max B_{v_i}(\mathbf{k}, \boldsymbol{\xi}) < 0) \vee (\min B_{v_i}(\mathbf{k}, \boldsymbol{\xi}) \geq 0)$ **then**
            Append j to $\mathcal{L}_{\text{elim}}$
        else
            Append j to $\mathcal{L}_{\text{save}}$
        end if
    end if
end for
return $\mathcal{L}_{\text{save}}, \mathcal{L}_{\text{elim}}$

Algorithm 12 [KZZV20, Algorithm 5] $\mathcal{L} = \text{Eliminate}(\mathcal{L}, \mathcal{L}_{\text{save}}, \mathcal{L}_{\text{elim}})$ (Parallel)

$n_{\text{save}} \leftarrow \text{length}(\mathcal{L}_{\text{save}})$
$n_{\text{elim}} \leftarrow \text{length}(\mathcal{L}_{\text{elim}})$
$n_{\text{replace}} \leftarrow n_{\text{elim}} - 1$
if $n_{\text{elim}} = 0$ **or** $\mathcal{L}_{\text{elim}}[1] > n_{\text{elim}}$ **then**
    return $\mathcal{L}$
end if
for $i \in \{1, \ldots, n_{\text{elim}}\}$ **do**
    if $\mathcal{L}_{\text{elim}}[i] \geq n_{\text{save}}$ **then**
        $n_{\text{replace}} \leftarrow i - 1$
        break
    end if
end for
parfor $i \in \{1, \ldots, n_{\text{replace}}\}$ **do**
    $\mathcal{L}[\mathcal{L}_{\text{elim}}[i]] \leftarrow \mathcal{L}[\mathcal{L}_{\text{save}}[n_{\text{save}} + 1 - i]]$
end parfor
return $\mathcal{L}$

Corollary 75. *[KZZV20, Corollarly 13] Suppose there exists a (feasible) global minimizer k^* of the chance-constrained program* (6.2)*. Then, while executing Algorithm 4, there always exists an item $(\mathbf{k}, B_J(\mathbf{k}), \Delta_{\text{lo}}) \in \mathcal{K}$ such that $k^* \in \mathbf{k}$.*

6.3 Application to Motion Planning

This section presents the application of CCPBA to trajectory optimization for mobile robots. I will use the reachable set computation methods described in Chapter 3, and the online planning methodology in Algrothim 1; however the obstacle representations will be probabalistic and Line 7 will be solved with CCPBA. Two key properties of RTD will be leveraged that make this approach attractive compared to the state-of-the-art. First, RTD represents reachable states of a robot executing parameterized trajectories as polynomial level sets. In traditional Model Predictive Control (MPC) methods, integration of the vehicle dynamic model is embedded in the trajectory optimization program by including the vehicle states as decision variables and representing the integration steps as equality constraints [GGL+12, PR14]. It can be computationally expensive to evaluate and solve such programs (see §5.1 and §5.2) . With RTD the decision variable space only consists of the trajectory parameters, and the constraints only consist of polynomial inequalities for collision avoidance; which improves computation time and allows the use of Algorithm 4. Second, the computation of the reachable states for RTD enables uncertainty in the robot model to be treated robustly. I will not analyse this aspect in the comparison in §6.4, but note that combining RTD with CCPBA allows one to treat uncertainty with the ego robot robustly, but other obstacle's probabilistically, a similar line of thinking to the approach in [FKBF+20].

The rest of the section is organized as follows. §6.3.1 provides examples of dynamics models associated with the robot and planned trajectories that are used in the results section. §6.3.2 explains how obstacles are represented and how the constraints for collision avoidance are constructed. §6.3.3 sets up the trajectory optimization program and solves an example motion planning problem for an autonomous vehicle performing lane changes with obstacle predictions that are normally distributed. §6.3.4 discusses how to use CCPBA with other probability distributions that are common in motion planning and robotics.

6.3.1 Dynamic Models

An autonomous vehicle will be the running example in this section; the same vehicle model will be used in the experiments in §6.4.

Example 76. *The inputs, $u \in U$, to the autonomous vehicle are an acceleration and wheel angle command. The dynamic model, $f_{\text{hi}} : \mathbb{R}_{\geq 0} \times Z_{\text{hi}} \times U \to \mathbb{R}^7$ is a single-track bicycle model with*

linear tire forces and small-angle assumptions for the slip angles.

$$\frac{d}{dt}\begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \\ v_x(t) \\ v_y(t) \\ \dot{\theta}(t) \\ \delta(t) \end{bmatrix} = \begin{bmatrix} v_x(t)\cos\theta(t) - v_y(t)\sin\theta(t) \\ v_y(t)\sin\theta(t) + v_y(t)\cos\theta(t) \\ \dot{\theta}(t) \\ u_1(t) \\ \frac{1}{m}(C_{y,f}\alpha_f(z_{\text{hi}}(t))\cos\delta(t) + C_{y,r}\alpha_r(z_{\text{hi}}(t))) - v_x(t)\dot{\theta}(t) \\ \frac{1}{I}(l_f C_{y,f}\alpha_f(z_{\text{hi}}(t))\cos\delta(t) - l_r C_{y,r}\alpha_r(z_{\text{hi}}(t))) \\ \frac{-1}{\tau_c}(\delta(t) - u_2(t)) \end{bmatrix} \tag{6.25}$$

$$\alpha_f(z_{\text{hi}}(t)) = \delta(t) - \frac{v_y(t) + l_f\dot{\theta}(t)}{v_x(t)}, \qquad \alpha_r(z_{\text{hi}}(t)) = \frac{-v_y(t) + l_r\dot{\theta}(t)}{v_x(t)}, \tag{6.26}$$

where m is the mass, I is the yaw moment of inertia, l_f and l_r are the distances from the front and rear axel to center of mass and C_y are the lateral cornerning stiffnesses. The vehicle parameters in the model are for a Chevrolet Equinox and can be found in [RKCL16, Table II].

The trajectory planning model is given below.

Example 77. *The autonomous vehicle from Example 76 tracks a trajectory parameterized by a quintic G^2 spline in lateral position [PB00] and constant longitudinal velocity. The parameters k_1 and k_2 refer to the longitudinal velocity and final lateral position, and will ultimately be the decision variables in* (6.2)*.*

$$\begin{bmatrix} x_{\text{des}}(t,k) \\ y_{\text{des}}(t,k) \\ \theta_{\text{des}}(t,k) \end{bmatrix} = \begin{bmatrix} k_1 t \\ \frac{10t^3k_2}{T^3} - \frac{15t^4k_2}{T^4} + \frac{6t^5k_2}{T^5} \\ \frac{30t^2k_2}{k_1T^3} - \frac{60t^3k_2}{k_1T^4} + \frac{30t^4k_2}{k_1T^5} \end{bmatrix}. \tag{6.27}$$

The initial and final yaw and curvature of the splines are 0, making this parameterization useful for planning lane change or lane-keeping maneuvers.

To execute a trajectory, the robot uses a proportional-derivative controller so the dynamic model can be written in closed-loop form (3.4).

6.3.2 Collision Function Computation

The this section summarizes the computation of a polynomial function whose 0 superlevel set identifies obstacles that could be in collision with the mobile robot. This function will be the constraint in (6.2), and the obstacle state will be the uncertain variable. The collision check will be enforced in 2D space as in Definition 12 and that there are finite number of obstacles as in Assumption 32. I next make assumptions about the space obstacles and the ego vehicle occupy.

Assumption 78. *Collision between the ego robot and obstacle is given by a function*

$$\mathcal{A}_{\text{col}} = \{(z, \zeta) \in Z \times Z_{\text{obs}} \mid h_{\text{col}}(z, \zeta) \geq 0\}, \tag{6.28}$$

where $\zeta \in Z_{\text{obs}}$ denotes the state of an obstacle. Such a function can be computed with Sums-of-Squares Polynomials for common robot and obstacle footprints. See Appendix A.3.

The collision set for the i^{th} obstacle at time t for the ego robot, with dynamics described by (3.4), executing a trajectory parameterized by k is then given by

$$\begin{aligned}\mathcal{X}_{\text{col},i} = \{(t, k, \zeta) \in [0, T] \times K \times Z_{\text{obs}} \mid \\ \exists\, z \in Z \text{ s.t. } (z, \zeta) \in \mathcal{A}_{\text{col},i},\ (t, z, k) \in \mathcal{Z}_{\text{TFRS}}\}.\end{aligned} \tag{6.29}$$

where $\mathcal{Z}_{\text{TFRS}}$ is defined as in (3.41) for the interval $[0, T]$. Using a modified version of (D_T) (see Appendix A) a polynomial function, $v_i : [0, T] \times Z_{\text{obs}} \times K \to \mathbb{R}$, can be computed; whose 0 superlevel set overapproximates the collision set:

$$(t, k, \zeta) \in \mathcal{X}_{\text{col},i} \implies v_i(t, k, \zeta) \geq 0. \tag{6.30}$$

The computation of v with Sums-of-Squares Programming is done offline; this means the set $\mathcal{A}_{\text{col}}$ must be defined offline. To deal with uncertain obstacle sizes, one could keep additional parameters (such as length and width) as part of the obstacle state; or store a library of polynomials for size ranges/shapes of obstacles. In §6.4.1.5 the later is used distinguish between car (rectangle) and pedestrian (circular) obstacles.

6.3.3 Online Optimization

The computation of v is performed offline. Online, assume:

Assumption 79. *A prediction module provides the trajectory planner with joint cumulative distribution functions, that satisfy Assumption 60, for realizations of the state of n_{obs} obstacles at a set of n_{pred} discrete timesteps.*

The online optimization program solved with Algorithm 4 is

$$\begin{aligned}&\min_{k \in K} J(k) \\ &\text{s.t. } \sum_{i=1}^{n_{\text{obs}}} \sum_{j=1}^{n_{\text{pred}}} \Pr(\{\xi_{i,j} \in \Xi \mid v_i(t_j, k, \xi_{i,j}) \geq 0\}) \leq \Delta,\end{aligned} \tag{6.31}$$

where $\xi_{i,j}$ indicates the realization of the states of the i^{th} obstacle at time t_j.

Consider the autonomous vehicle from Examples 76 and 77 with a time horizon of $T = 4$ s and an initial condition ranges for position of $\pm$ 0.05 m, heading $\pm$ 0.005 rad, velocity 12-13 m/s and wheel angle $\pm$ 0.005 rad. The parameter ranges, K, are $k_1 \in [10, 16]$ ms for velocity and $k_2 \in [-1, 5]$ m for final lateral position. The collision set is represented with a degree 8 polynomial as in (6.30). The vehicle is driving on a straight two-lane road and wants to either plan a lane change or lane keeping maneuver; which can be accomplished by the following the cost function:

$$J(k) = k_1^2 - 26k_1 + 0.30k_2^4 - 2.09k_2^3 + 3.43k_2^2 + 3 \tag{6.32}$$

This cost function has a global minimum at $k = (13, 3.7)$ and a local minimum at $k = (13, 0)$. The obstacle state is given by the 2D position center of mass, i.e. $\zeta \in \mathbb{R}^2$, and its footprint is assumed to be a rectangle of width 2.5 m and length 5 m. The obstacle predictions are given as normal distributions

$$\xi_{i,j} \sim \mathcal{N}(\hat{\xi}_{i,j}, \Sigma_{i,j}). \tag{6.33}$$

and generated by linearizing a kinematic bicycle model and propagating the covariances forward similar to an Extended Kalman Filter [Thr02, (3.52)]. The normal distribution has an computational advantage in that the coordinate transform

$$\xi_{i,j} \leftarrow \hat{\xi}_{i,j} + \text{chol}(\Sigma_{i,j})\xi_{i,j}, \tag{6.34}$$

can be applied to the ξ component of each constraint, then the joint cumulative distribution function can be written analytically as

$$F(\boldsymbol{\xi}) = \prod_{r=1}^{2} \frac{1}{2}\left(\text{erf}\left(\frac{\overline{\xi}_r}{\sqrt{2}}\right) - \text{erf}\left(\frac{\underline{\xi}_r}{\sqrt{2}}\right)\right). \tag{6.35}$$

The threshold for chance constraint violation is set to $\Delta = 0.01 - n_{\text{pred}}(1 - F([-5, 5]^2))$, which corresponds to integrating over 5 standard deviations, as in Remark 62.

Figure 6.2 shows the performance of CCPBA as it solves the trajectory optimization problem depicted in Figure 6.1. CCPBA progresses through iterations until an optimality tolerance of $\epsilon = 1.5$ is met. Since Algorithm 4 returns a subbox $\mathbf{k}^*$, a solution, k^*, is found by taking the midpoint in each dimension. CCPBA finds a feasible, suboptimal solution at iteration 11, corresponding to driving straight; it then improves the cost as patches in $K \times \Xi$ are refined. For this example, the run time of Algorithm 1 on a computer with a 48 GB Nvidia Quadro RTX 8000 GPU, 3.60 GHz

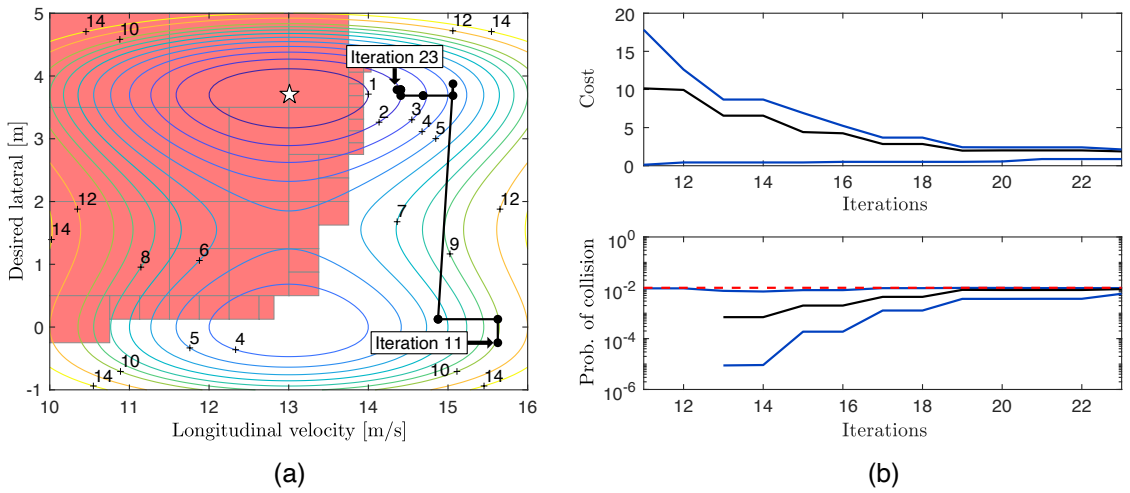


Figure 6.2: Subfigure (a) shows contours of the cost function (6.32) in the decision variable space K with blue indicated lower cost regions, and yellow higher. The white star indicates the global minimum. CCPBA solves the chance-constrained program (6.31) with the obstacle depicted in Figure 6.1. The black line shows the cost of the solution $J(k^*)$ as CCPBA progresses from iteration 11 (first feasible) to 23, when an optimality tolerance of $\epsilon = 1.5$ was reached. The red patches indicate infeasible regions of the decision variable space identified by iteration 23. Subfigure (b) shows the cost in black and lower and upper bounds from CCPBA, J^*_{lo} and J^*_{up}, at each iteration (blue). The bottom plot shows the probability of constraint violation using 1e6 Monte Carlo points per timestep (black) and its lower and upper bounds Δ_{lo} and Δ_{up} (blue). The algorithm finds a feasible, suboptimal solution at iteration 11, and improves the cost as patches in $K \times \Xi$ are refined. The run time for this example was 0.22 s.

Intel i9 CPU, and 128 GB of RAM was 0.22 s.

6.3.4 Non-Gaussian Predictions

In the example in §6.3.3, uncertainty in the chance constraints was normally distributed. This meant F in Assumption 60 could be defined by using a coordinate transform (6.34) and the `erf` function (6.35). For CCPBA it is preferable to have a joint cumulative distribution function, F, that can be expressed analytically for a subbox, to improve computation time and avoid errors associated with numerical integration.

However, one may wish to use a prediction algorithm where the obstacle distributions are not Gaussian. In this section I discuss two types of representations that are common in motion planning literature and how one could construct F analytically for each of them.

6.3.4.1 Particle-based

Particle-based approaches, are a common method to represent probability densities of uncertain variables in robotic applications. For example [AVJR21] uses a particle-based approach to represent the future position and velocities of vehicles on a highway. In these approaches the probability density $p : \Xi \to \mathbb{R}_{\geq 0}$ of the uncertain parameter is represented by a collection of n particles

$$p(\xi) = \sum_{i=1}^{n} q_i \delta_0(\xi - \hat{\xi}^{(i)}) \tag{6.36}$$

where q_i are positive weights that sum to 1, δ_0 is the dirac delta, and $\hat{\xi}^{(i)}$ centers the dirac delta for the i^{th} particle. A joint cumulative distribution function, F, can be found for the particle representation (6.36) in two steps. First, a Kernel Density Estimator (KDE) can be applied to give a smoothed estimate of the probability density

$$\hat{p}(\xi) = \sum_{i=1}^{n} q_i p_H(\xi - \hat{\xi}^{(i)}), \tag{6.37}$$

where p_H is a kernel with bandwidth H. The choice of kernel is heuristic, but to satisfy Assumption 60 and enable fast execution of CCPBA, one should pick one that can be integrated analytically; for example, a Gaussian kernel

$$p_H(\xi - \hat{\xi}^{(i)}) = (2\pi^{-\frac{m}{2}})\text{det}(H)^{-\frac{1}{2}} e^{-\frac{1}{2}(\xi - \hat{\xi}^{(i)})^\top H^{-1}(\xi - \hat{\xi}^{(i)})}, \tag{6.38}$$

where the bandwidth H is a diagonal matrix. See [KLS11] for an algorithm to compute H. Then the joint cumulative distribution function can be determined with the `erf` function as follows

$$F(\boldsymbol{\xi}) = \sum_{i=1}^{n} q_i \prod_{r=1}^{m} \frac{1}{2} \left(\text{erf}\left(\frac{\overline{\xi}_r - \hat{\xi}_r^{(i)}}{\sqrt{2H_{r,r}}} \right) - \text{erf}\left(\frac{\underline{\xi}_r - \hat{\xi}_r^{(i)}}{\sqrt{2H_{r,r}}} \right) \right). \tag{6.39}$$

6.3.4.2 Occupancy grids

Occupancy grids arise from prediction algorithms that that rely on discretizing the space Ξ, such as the Markov chain model in [TFS19] or the confidence-aware approach in [BHFK$^+$19, FKBF$^+$20]. In these approaches the cumulative distribution can be thought of as a collection of weighted uniform distributions. Let Ξ be the unit box $[0, 1]^m$ as in Assumption 65. Given a multi-index L, for the number of cells in each dimension, the joint cumulative distribution in a subbox $\boldsymbol{\xi}$ can

be given by

$$F(\boldsymbol{\xi}) = \sum_{I \leq L} q_I \prod_{r=1}^{m} \frac{\max(b(\boldsymbol{\xi}, i_r, l_r) - a(\boldsymbol{\xi}, i_r, l_r), 0)}{2^{1-l_r}} \tag{6.40}$$

$$a(\boldsymbol{\xi}, i_r, l_r) = \max(\underline{\xi}_r, 2^{1-l_r} i_r - 2^{-l_r}) \tag{6.41}$$

$$b(\boldsymbol{\xi}, i_r, l_r) = \min(\overline{\xi}_r, 2^{1-l_r} i_r + 2^{-l_r}), \tag{6.42}$$

where I is a multi-index (with i_r ranging from 1 to l_r) identifying the cell.

6.4 Results

This section provides a simulation comparison of CCPBA against two chance-constrained Model Predictive Control (MPC) algorithms in §6.4.1. §6.4.2 describes a hardware demonstration where the Segway from Figure 3.2b avoids a moving obstacle.

6.4.1 Lane Change Comparison

This section provides results for a comparison of CCPBA against two chane-constrained MPC algorithms, for a lane-changing problem with the autonomous vehicle in Example 76. §6.4.1.1 describes details of the problem formulation that apply to all planners. §6.4.1.2 describes details of the CCPBA implementation. §6.4.1.3 describes the implementation of the MPC algorithms.

In the first experiment, §6.4.1.4, has one car obstacle in the oncoming lane, and vary the noise in its prediction. The algorithms' performance is compared in terms of cost and run time. In the second experiment, §6.4.1.5, multiple obstacles are introduced, increasing the number of constraints, and the algorithms' time to find a feasible solution is compared. All experiments in this section are run on a desktop computer with a 48 GB Nvidia Quadro RTX 8000 GPU, 3.60 GHz Intel i9 CPU, and 128 GB of RAM.

6.4.1.1 Problem setup

The simulations use the Equinox model described in Example 76. The initial condition ranges for are a position of $\pm$ 0.05 m, heading $\pm$ 0.005 rad, velocity 12-13 m/s and wheel angle $\pm$ 0.005 rad. For the cost in Experiment 1, a desired trajectory using the reference model (6.27) in Example 77 with the desired parameters $k_{1,\text{des}} = 13$ m/s and $k_{2,\text{des}} = 3.7$ m is used.

For the obstacles, the state of an obstacle, $\zeta \in \mathbb{R}^2$, is defined by Cartesian coordinates of its center of mass. The footprint of a car obstacle is a rectangle of width 2.5 m and length 5 m, with 0 rad heading (in line with the road). The footprint of a pedestrian obstacle is a circle of radius

0.5 m. The probability of the state of the surrounding obstacles is given as normal distributions (6.33), as described in §6.3.3. The predictions are given at a time discretization of 4 hz and all of the planners perform a collision check of a 4 s time horizon.

6.4.1.2 CCPBA implementation

The trajectory parameterization is the same as in Example 77. The parameter ranges, K, are $k_1 \in [10, 16]$ ms for velocity and $k_2 \in [-1, 5]$ m for final lateral position. The collision sets are represented with a degree 8 polynomials, as in (6.30). Since the obstacles are normally distributed the coordinate transform (6.34) and (6.35) is used to evaluate the probability mass in each subbox. The cost function used in the online optimization is a quadratic cost function that minimizes the distance of k to desired parameters k_{des}.

$$J(k) = q_1(k_1 - k_{1,\text{des}})^2 + q_2(k_2 - k_{2,\text{des}})^2 \tag{6.43}$$

where $q \geq 0$ are positive weights. This choice of cost function is motivated by the comparison to MPC algorithms, as it is common for real-time MPC implementations to minimize a similar squared distance to a reference trajectory [GZQ+20a]. In §6.4.1.3 formulates such a cost function for the reference trajectory parameterized by $k_{1,\text{des}}$ and $k_{2,\text{des}}$. CCPBA is also able to leverage efficiencies related to cost functions of this form.

Remark 80. *The minimum value of, and the minimizer of the cost function* (6.43) *over a given subbox* $\mathbf{k}$ *can be determined analytically. This offers two major computational benefits. First, the cost function does not need to be projected onto the Bernstein basis, saving memory and computations associated with subdivision of Bernstein patches. Second, convergence of Algorithm 4 is improved, since once an item is identified as feasible by Definition 68, the best cost and minimizer for the subbox can be determined immediately. This means the best solution can be stored and the item can be added to the elimination list in Algorithm 10. For the experiments in this section and the hardware demonstration in §6.4.2 a modified version of Algorithm 4 is used, where these efficiencies are leveraged.*

To compare the cost for CCPBA to the MPC algorithms, I took the returned parameters k^* and forward integrate the Equinox tracking the trajectory using `ode45`. Then the state of the trajectory is evaluated on the cost function used by the MPC algorithms. This method for comparing cost is favorable to the MPC algorithms, since they plan with the full vehicle model.

6.4.1.3 MPC implementations

CCPBA is compared to Model Predictive Control algorithms using Cantelli's inequality [WJW20] and the Chernoff bound [ZK14]. The MPC algorithms use the bicycle model of the Equinox, integrated with rk4 integration, at 4 hz. Constraints enforce input limits acceleration in $[-8, 3]$ ms^{-2} and wheel angle in $[-0.5, 0.5]$ rad. Additionally a friction ellipse constraint limits the combined lateral and longitudinal acceleration to 9.81 ms^{-2}.

Obstacles are represented as point obstacles their 2-D center of mass coordinates and collision checked using an ellipse. The ellipse is given by the 0 superlevel set of the quadratic equation:

$$h_{\text{col}}(z, \zeta) = 1 - \left(\zeta - \begin{bmatrix} x \\ y \end{bmatrix} \right)^\top R(\theta) \begin{bmatrix} \frac{1}{a^2} & 0 \\ 0 & \frac{1}{b^2} \end{bmatrix} R(\theta)^\top \left(\zeta - \begin{bmatrix} x \\ y \end{bmatrix} \right), \tag{6.44}$$

where $R : \mathbb{R} \to \mathbb{R}^{2\times 2}$ is the 2D rotation matrix. The major and minor axis lengths are $a = 6.9$ m and $b = 2.95$ m for the car obstacle and $a = 3.7$ m and $b = 1.7$ m for the pedestrian obstacle. Chance-constraints are represented with Cantelli's inequality (6.45), as in [WJW20],

$$\Pr(\{\xi \in \Xi \mid h_{\text{col}}(z, \xi) \geq 0\}) \leq \frac{\text{Var}[h_{\text{col}}(z, \xi)]}{\text{Var}[h_{\text{col}}(z, \xi)] + \mathbb{E}[h_{\text{col}}(z, \xi)]^2}, \tag{6.45}$$

or the Chernoff bound, as in [ZK14]:

$$a \ln(\mathbb{E}[\exp(a^{-1} h_{\text{col}}(z, \xi))]) \leq a \ln \Delta,\ a > 0 \implies \Pr(\{\xi \in \Xi \mid h_{\text{col}}(z, \xi) \geq 0\}) \leq \Delta. \tag{6.46}$$

Boole's inequality is applied as follows

$$\begin{aligned} \Pr(\{\xi_{i,j} \in \Xi \mid h_{\text{col}}(z, \xi_{i,j}) \geq 0\}) &\leq \Delta_{i,j}, \\ \sum_{i=1}^{n_{\text{obs}}} \sum_{j=1}^{n_{\text{pred}}} \Delta_{i,j} &\leq \Delta, \end{aligned} \tag{6.47}$$

where $\xi_{i,j}$ indicates the realization of ξ for the i^{th} obstacle at the j^{th} timestep, and $\Delta_{i,j}$ are added to the MPC progrma as decision variables. The Chernoff approach requries numerical integration, and uses 1e4 monte carlo points, which should yield error of approximately 1% [AG07]. However the exponential over approximates the indicator function, so the risk bounds are typically conservative. Unfortunately, proving a lower bound on how conservative they are is difficult [NS07]. The

cost function for the MPC planners is

$$J(z,u) = \sum_{j=0}^{n_{\text{pred}}} q_1(z_{4,j} - k_{1,\text{des}})^2 + q_2(z_{2,\text{j}} - y_{\text{des}}(t_j, k_{\text{des}}))^2 \tag{6.48}$$

where $z_{2,j}$ and $z_{4,j}$are the lateral position and longitudinal velocity at timestep j. The parameters k are the desired velocity and final lateral position, and y_{des} is from Example 77 (6.27). The compared cost for the MPC algorithms is simply that of the solution returned by IPOPT. This is again favorable to the MPC algorithms, since I do not check how accurately the dynamics of the vehicle are captured in the solution.

To solve the MPC programs, I used Casadi [AGH+18b] with its default settings for IPOPT. Note that this implementation is different from the solvers used in [WJW20, ZK14] so run times may differ; however the main focus of this comparison is to see how their methods for evaluating chance constraints compare to CCPBA in terms of feasibility and cost.

6.4.1.4 Experiment 1: Varying Process Noise

The first experiment considers left lane change scenario with a single oncoming obstacle in Figure 6.1. The range of initial conditions for the ego vehicle, decision variable space, and cost function for CCPBA is the same as the example in §6.3.3. The obstacle's predictions are generated using a Dubin's car model with a proportional controller tracking a desired lateral position, and 0 acceleration. Predictions are generated for a time horizon of $T = 4$ s and 4 hz, using the same method one finds in an Extended Kalman Filter [Thr02, (3.52)]. At each timestep introduce process noise with the covariance

$$\Sigma = M\text{diag}([0,\ 0.25,\ 0,\ 0.75]), \tag{6.49}$$

where $M > 0$ is a multiplier that is adjusted to increase uncertainty in the prediction. The probability threshold Δ is set to 0.01 for all planners. For CCPBA the optimality tolerance is set to $\epsilon = 1$.

The parameters of the simulation are varied over 1,000 trials. The ego vehicle starts at a random initial condition selected from the range used to compute the reachable set. The initial mean condition for the obstacle is $[-20; 3.7; 0; 14.5]$ plus a uniformly distributed offset of $\pm$[1;0.25;0.005;0.5]. The desired lateral position of the obstacle is 3.7 $\pm$ a uniformly distributed offset of 0.25 m. The initial covariance for each the obstacle states is 1e-6. The process noise, M, takes one of 50 values logarithmically spaced between 1e-4 and 1. In terms of position uncertainty $T = 4s$, $M = 1e-4$ roughly corresponds to standard deviations of approximately 0.075 and 0.05 m at in the x and y

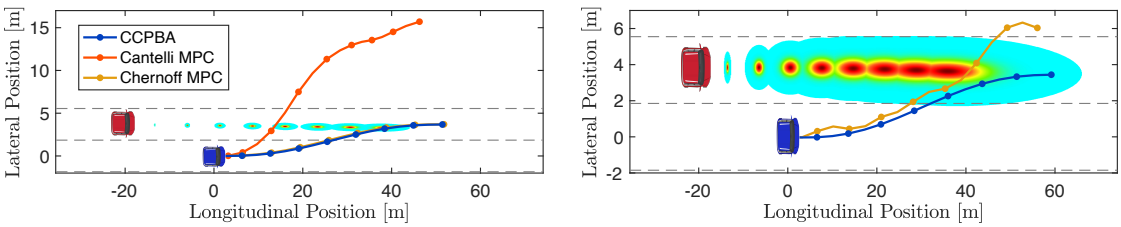


Figure 6.3: Planned trajectories from each planner in Experiment 1. The probability threshold for the chance-constraint is 1%. Trajectories planned by CCPBA, Cantelli MPC, and Chernoff MPC are plotted in blue, orange, and gold. The colormap shows the probability density for the obstacle at each timestep, plotted at 2 hz for clarity, with red being more likely outcomes. The blue edge corresponds to confidence ellipsoids containing 5 standard deviations. Subfigure (a) shows the scenario with the largest uncertainty, M in (6.49), where Cantelli MPC found a feasible solution. Subfigure (b) shows the scenario with the largest uncertainty where CCPBA was able to complete the lane change; defined as having a final lateral position within 0.7 m of lane center.

coordinates $M = 1$ corresponds to standard deviations of approximately 7.5 and 0.5 m.

The final cost and solve time of the Cantelli, Chernoff, and CCPBA algorithms are compared in Figure 6.4. The cost for MPC with Cantelli's inequality increases rapidly at relatively low uncertainty. It is unable to find feasible solutions at $M = 0.03$, which roughly corresponds to a standard deviation of 0.1 m at $T = 4$ s. On the other hand Chernoff MPC and CCPBA are able to find feasible solutions for all values of M, but the cost of CCPBA compared to Chernoff MPC is on average 88% higher. However when looking at lane changes completed, defined as the final lateral position of the trajectory being between within 0.7 m of target lane center; CCPBA is able to complete 932 lane changes, compared to 899 for Chernoff MPC, and 433 for Cantelli MPC. The fact that CCPBA is able to complete 4% more lane changes than Chernoff MPC suggests that, despite having a higher cost on average, it is able to better or comparably accomplish its objective. Additionally the run time of CCPBA is on average 3% of that of Chernoff MPC. As the uncertainty increases, the solve times for both algorithms increase with CCPBA platueing at 1.2 s; when the GPU runs out of memory. In these instances CCPBA is still able to return a feasible solution, although it has not met the optimality tolerance. CCPBA is able to make the lane change until $M \approx 0.5$, which corresponds to a standard deviation of roughly 0.35 m at $T = 4$ s. Figure 6.3 shows the trajectories planned when Cantelli's inequality is unable to find a feasible solution, and at the largest M value where CCPBA is able make the lane change ($|y_{\text{des}} - 3.7| \leq 0.7$ m).

6.4.1.5 Experiment 2: Feasibility Multiple Obstacles

This section comapres how the CCPBA compares to the Chernoff MPC algorithm in terms of the time it takes to find feasible solutions as the number of constraints increases. The ego vehicle uses

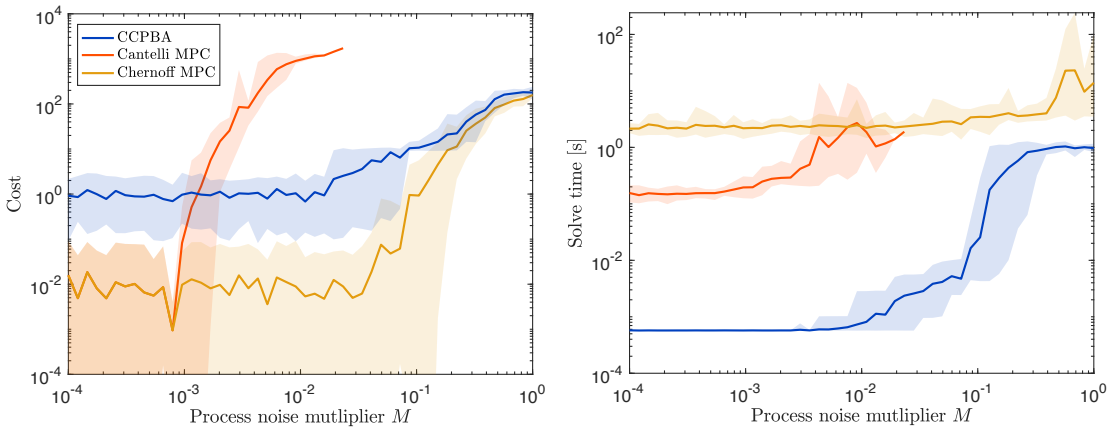


Figure 6.4: Cost and solve time for 1,000 simulations with varying process noise and initial conditions, for the lane change scenario in Figure 6.1. Lines show the mean, and the faded regions cover the max/min values. CCPBA is plotted in blue. MPC algorithms using Cantelli's inequality and the Chernoff bound are plotted in orange and gold. Cantelli's inequality is not able to find feasible solutions for high uncertainty scenarios.

the same trajectory model, initial condition, and cost function as the first experiment; however there are two vehicles in each lane and two pedestrians, one of which is jaywalking in the street. The vehicle obstacles are generated with similar dynamics and process noise as the one in §6.4.1.4, with $M = 0.1$. The pedestrian obstacles have constant heading dynamics, a average speed of 1 ms^{-1}. For this comparison I only compared CCPBA to the Chernoff MPC, since from §6.4.1.4, Cantelli's inequality struggles to find reasonable solutions even in the one obstacle case. The cost of both algorithms is set to $J(\cdot) = 0$.

Figure 6.5 shows the experimental setup, and solution trajectories found with all 6 obstacles considered. To see how the performance is affected as the number of constraints is increased, obstacles are introduced one at a time, in the order indicated in Figure 6.5. Similar to Experiment 1, the initial conditions of each obstacle are varied for 1,000 trials. The results are shown in Figure 6.6. There were 125 trials with 6 obstacles where CCPBA was not able to find a feasible solution; however this is expected as CCPBA is planning over a lower-dimension trajectory space. As the number of obstacles increases, the median time the Chernoff MPC takes to find a feasible solution increases; from 1 s with 1 obstacle to 16 s with 6 obstacles. The time for CCPBA increases as well, but the values are much lower overall– 2 ms with 1 obstacle to 0.25 s with 6 obstacles. This shows promise for the implementation of CCPBA in a receding horizon motion planner, since it is able to quickly return suboptimal, but feasible solutions.

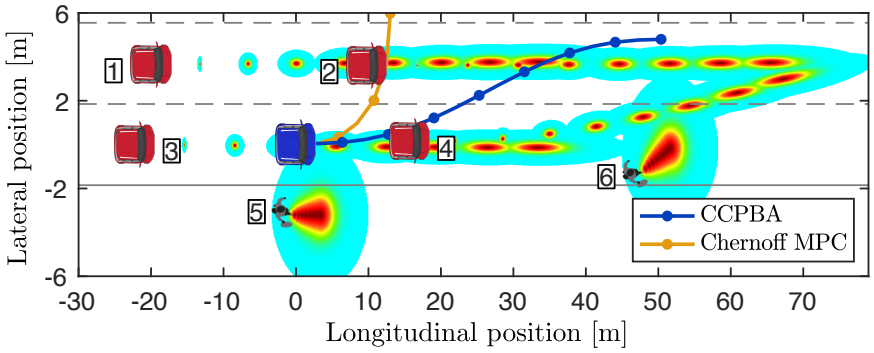


Figure 6.5: Scenario for Experiment 2. The ego vehicle is plotted in blue. The probability threshold is 1%. The solution trajectories for CCPBA (blue) and the Chernoff (gold) method for a trial with all obstacles is plotted. Car obstacles are plotted in red, pedestrians in gray. Their probability densities, plotted at 2 hz for clarity, are colored cyan to red, with red indicating more likely outcomes. The confidence ellipsoids containing 5 standard deviations for each density is plotted. The obstacle numbering indicates the order in which they are cumulatively introduced.

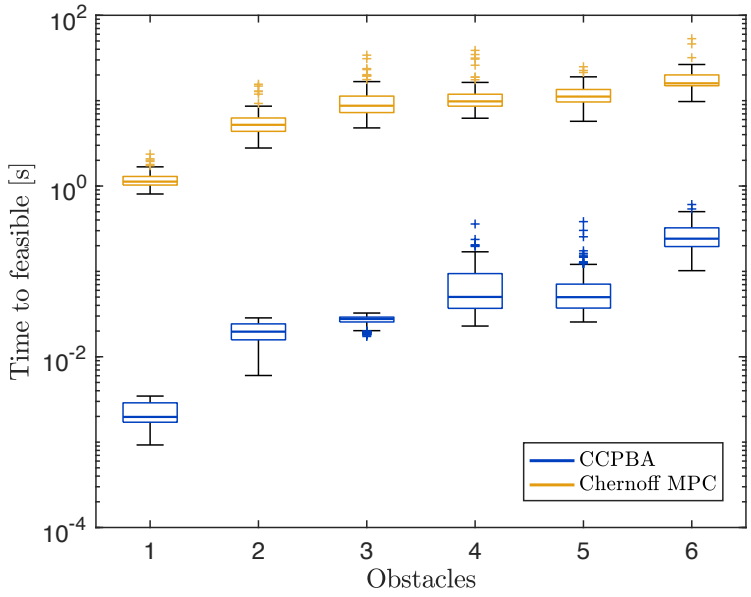


Figure 6.6: Box-and-whisker plots for CCPBA (blue) and Chernoff MPC (gold) time to return a feasible solution for 875 simulations, with obstacles cumulatively introduced as indicated in Figure 6.5. The solve times for CCPBA are roughly 2 orders of magnitude less than the Chernoff MPC. There were 125 simulations (not plotted) where Chernoff MPC found a feasible solution and CCPBA did not.

6.4.2 Hardware Demonstration

This section summarizes the hardware demonstration of CCPBA, where the Segway from Figure 3.2a avoids a manually controlled RC car. A constant velocity, k_1, and yaw rate, k_2, trajectory parameterization is used as in Example 4; with ranges of [0,1] m/s and [-1,1] rad/s, and the commanded change in yawrate is limited to 1 rad/s per planning iteration. The time horizon of the planned trajectory is $T = 1$ s, and the collision checking is performed at 4 hz.

The Segway plans in a receding-horizon loop operating at a frequency of 2 hz as in Algorithm 1. The cost function at each planning iteration is the quadratic cost (6.43). The desired longitudinal velocity and yawrate are generated by a pure pursuit controller [Cou92], tracking a coarse path generated by a higher-level planner using Dijkstra's algorithm to find the shortest obstacle free path to the specified goal. An optimality tolerance of $\epsilon = 0.001$ is used. If Algorithm 4 is unable to find a feasible solution, the Segway executes a braking maneuver by commanding 0 velocity and yaw rate.

For sensing the Segway uses a planar Hokuyo UTM-30LX LIDAR and Vectornav VN-100 IMU for mapping with Google Cartographer [HKRA16]. The Segway runs ROS (Robot Operating System) for communication between the mapping, obstacle detection, and trajectory optimization modules. All computations are run onboard, on a computer with a 4 core 2.90 GHz Intel i7 processor, 64 GB RAM, and an 8GB Nvidia Geoforce GTX 1080 GPU. A publicly available obstacle detection package that specializes in the detection of rectangular shaped objects[1] is used. The package estimates the obstacle's state as 2D position $(x, y) \in \mathbb{R}^2$ and linear velocity $(\dot{x}, \dot{y})$. The predictions of the obstacles are generated with the constant velocity model

$$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}_{j+1} = \begin{bmatrix} 1 & 0 & \tau_s & 0 \\ 0 & 1 & 0 & \tau_s \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}_j , \tag{6.50}$$

and propagate a normal distribution forward with a sample time $\tau_s = 0.25$ s and process noise of `diag`([1e-6 1e-6 1e-6 1e-6]). Uncertainty is introduced, by varying the initial covariance of the velocity. Since there is not an accurate estimate of the obstacle's orientation, it is conservatively treated as an ellipse; hence the sample space in the evaluation of the chance constraints, (6.3), is simply the 2D position of its center. The domain of integration, $\Xi \subset \mathbb{R}^2$, is considered to be to be 5 standard deviations from the mean of each normal distribution, using the coordinate transform (6.34), and address the probability mass outside of this domain as in Remark 62. The implementation of CCPBA is performed on the GPU using MATLAB's CUDA toolbox to generate

[1] https://github.com/kostaskonkk/datmo

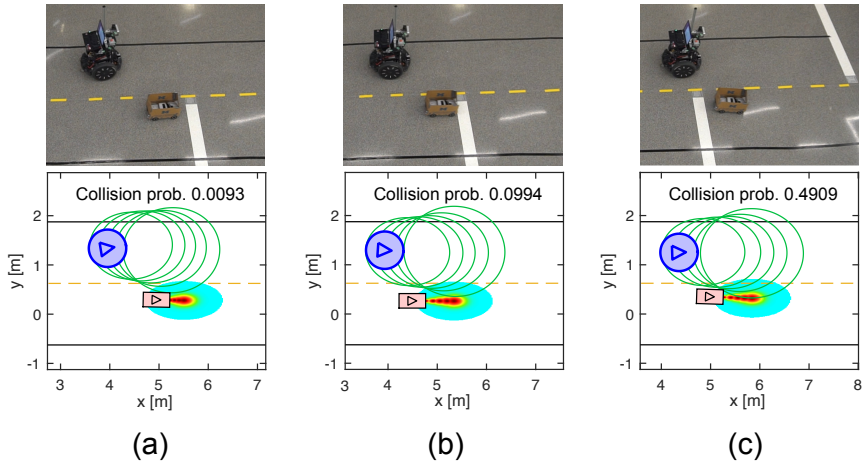


Figure 6.7: Snapshot of a trial from the hardware demonstration where the Segway robot (planning with CCPBA) attempts to lane keep while avoiding a manually controlled RC car with a probability thresholds Δ =0.01, 0.1, and 0.5 for subfigures a, b, and c. A snapshot of the planner view for each is shown at the bottom. The blue circle and arrow shows the current state and orientation of the Segway. The reachable sets (0 level sets of $v(t_j, k^*, \cdot)$) for the next planning iteration are shown in green. The light red rectangle shows the current position of the RC car. The probability density of the obstacle, generated with (6.50), is plotted from cyan (less likely) to red (more likely).

the required functions. The initial setup and projection onto the Bernstein basis (Algorithm 4 Line 1) is performed in MATLAB.

In the demonstration the Segway is first tasked with performing a lane keeping task where is changes lanes and drives adjacently behind the RC car. The risk thresholds, Δ, are varied from 0.5, 0.1, and 0.01. The initial covariance used in these trials is `diag`([1e-6 1e-6 0.01 0.0025]). A snapshot from one run is for each risk threshold shown in Figure 6.7. Unsurprisingly the Segway keeps further distance from the RC car as the risk threshold is decreased. Then a series of trials where the RC car starts at the other side of the course and veers into the Segway's lane is performed. The initial covariance used in these trials is `diag`([1e-6 1e-6 0.005 0.0025]). The distance between the Segway and the obstacle increases as the risk threshold decreases. Figure 6.8 shows a box and whisker plot of the run time of Algorithm 4 plotted as a function of the threshold for collision probability, Δ. The variance of the run time decreases as the risk threshold increases, since it is easier for Algorithm 9 to find feasible solutions; however the expected run time is similar for all thresholds. The probability threshold is then set to $\Delta = 0.1$ and the Segway completes several trials driving back and forth between 2 goal points with a randomly moving obstacle. The initial covariance used in these trials is `diag`([1e-6 1e-6 0.01 0.01]). The demonstration shows that CCPBA can be implemented for real-time receding horizon motion planning.

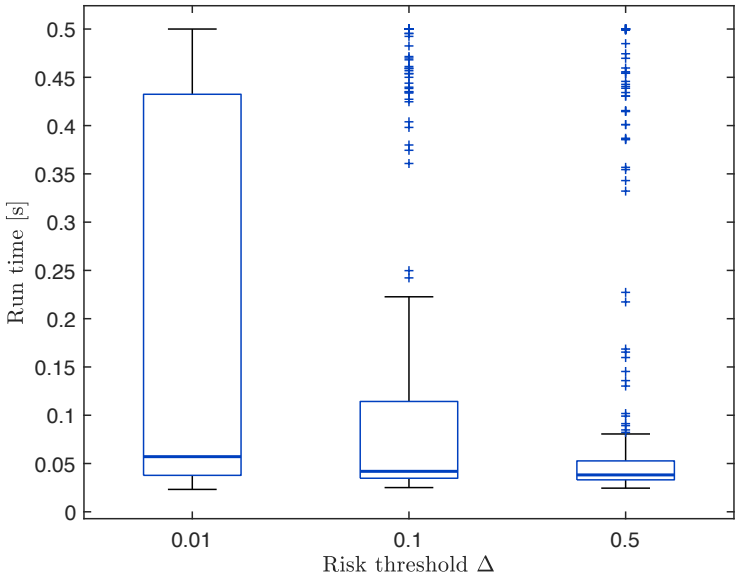


Figure 6.8: Run time of Algorithm 4 plotted for each risk threshold Δ in the hardware demonstration. The stopping criteria was an optimality tolerance of $\epsilon = 0.001$ or a time limit of 0.5 s. Only times where an obstacle was detected and CCPBA found a feasible solution are shown. The variance of the run time decreases as the risk threshold increases, since it is easier for Algorithm 9 to find feasible solutions; however the expected run time is similar for all thresholds.

6.4.3 Discussion

CCPBA has several advantages. First it will find a global minimum if a feasible one exists. It does not require an initial guess and does not converge to local minima. Additionally the only restriction on the chance constraints are that they are polynomial, and the probability mass in a box can be evaluated. There are no required assumptions about convexity or the uncertainty vector appearing affinely that are common in chance-constrained literature. Additionally,the subdivision parameters and direction choice, are the only hyper parameters that need to be tuned. Furthermore, CCPBA, can be modified to run as an anytime algorithm, where the current best solution is returned when a time limit is reached. This makes it especially useful for motion planning applications, as it typically finds feasible solutions quickly. For example in the lane change problem in §6.3.3, CCPBA quickly returns a feasible solution of driving straight, then it determines a lane change is feasible after further refinement.

The disadvantage of CCPBA is that the decision variable space and uncertainty vector space are limited, and the number of patches and associated computations increases exponentially with dimension. In the presented example, there is a total dimension (decision variables plus uncertain variables) of 4. The computations may be improved over the results in this paper, by exploiting the structure of the projection and subdivision matrices; however the major bottleneck is the number

of patches. To this end, future research on efficient splitting strategies to evaluate the chance constraints, possibly drawing on theory from adaptive, numerical integration [DR07] should be conducted.

CHAPTER 7

Adaptive Planning and Control

This chapter presents an adaptive, stochastic approach to dealing with uncertainties in a mobile robot's dynamics. Although the reachable set computation in Chapter 3, provides robust guarantees with respect to the ego vehicle's dynamics; the computation of the reachable set has to take place offline. This chapter proposes adaptive, chance-constrained methods for planning and control that take into account an online estimate of the tire force and uncertainty. This section focuses on the example of an autonomous vehicle driving on varying road surfaces, however I stress that the same techniques can be applied to a wide variety of robot models.

§7.1 describes related work focusing on estimation and control of vehicles driving on varying road surfaces. §7.2 explains the vehicle models that will be used throughout this section. §7.3 details two state-of-the art algorithms for estimating vehicle tire forces in linear and nonlinear regions. §7.4 details a method for real-time chance-constrained Model Predictive Control (MPC) problems and proposes the main contribution of this chapter: an algorithm to utilize the tire force estimates online in an adaptive framework [VQB21]. §7.5 compares the proposed algorithm to adaptive and deterministic approaches. The work in this chapter was completed in collaboration with Karl Berntorp and Rien Quirynen at Mitsubishi Electric Research Labs.

7.1 State-of-the-art

Control systems for autonomous vehicles actuate the vehicle through tire–road contact; therefore knowledge of the tire–road relation is of high importance. The interaction between tire and road is highly nonlinear, and the parameters describing the nonlinear relation vary heavily based on the road surface and other tire properties [Sve07, Gus97]. In normal to moderate driving conditions, vehicle tire forces are often modeled as a linear function of the slip angles (lateral) and slip ratios (longitudinal) produced at each wheel [Raj11]. The coefficient for the slope of such curves is referred to as the *stiffness*, specifically the *cornering stiffness* in the case of lateral forces. Knowledge of the tire stiffness can be used directly in Advanced-driver Assistance Sys-

tems (ADAS) [DCTBB12], and can be of particular importance in events where the vehicle needs to plan aggressive or safety-critical maneuvers [WSE18]. Additionally partial knowledge of the tire stiffness can be used to classify surface types for road-condition monitoring [APT12, Gus97].

Model Predictive Control (MPC) has been effectively applied to several automotive control applications [FBA+07, DCKB16, QBDC18]. MPC solves an optimization problem, where a dynamic model of the vehicle is integrated over a fixed time horizon to minimize a user-specified cost subject to constraints on the inputs and states. In many ADAS applications, the MPC program is nonlinear (NMPC) due to the vehicle model and constraints. NMPC is often solved with sequential quadratic programming (SQP), where a tailored convex solver is used to solve a sequence of structured quadratic programs (QPs) [GZQ+20b]. In recent years, many such algorithms have been developed to exploit particular sparsity structures that arise in SQP based NMPC, such as the recently proposed QP solver in [QDC20] and references therein. In [FDCQ20], an optimization algorithm is proposed for stochastic NMPC (SNMPC), which uses a tailored Jacobian approximation along with an adjoint-based SQP method. This SNMPC formulation considers individual chance constraints that are approximated using online linearization-based covariance propagation equations [TVC+15a].

Since the performance of MPC depends heavily estimating the tire parameters correctly, recent studies have focused on adaptive controllers, where uncertain parameters are estimated and the model is updated online. In [BZTB18], a robust MPC formulation is proposed, where a parameter associated with the steering offset is estimated. In [LCZW19, CLL14], least-squares algorithms are used to estimate the cornering stiffness and road friction, which are utilized in the MPC model and constraints. However, these two works do not consider uncertainty of the estimated stiffness, and all use linearized vehicle models.

In addition, even in the linear tire region, a difficulty when learning the tire-friction function using automotive-grade sensors is that the amount of sensors is limited, and they are relatively low grade [Gus09]. Moreover, not only do the sensors only provide indirect measurements of the friction, they do not even measure some vehicle states, such as the lateral velocity; which is important for learning the tire friction. In [BDC18], a particle-filter based algorithm is proposed, which estimates the mean and covariance of the tire stiffness using data from commonly available inertial sensors. [BQUDC19] utilized this cornering-stiffness estimator in NMPC by selecting from a library of predefined Pacejka tire models. Unfortunately performance and safety may be compromised with this approach, since there will be mismatch between the tire model and reality.

Obtaining data for the nonlinear regions of the tire curve is challenging as it requires to drive the vehicle to the limits of its performance envelope. Furthermore, driving in the nonlinear region of the tire force function before a reliable model of the same is obtained is challenging and possibly dangerous. For collision-imminent steering [WDSE20] proposes an adaptive approach

where an Unscented Kalman Filter estimates the coefficient of friction in a Pacejka model. They show that this greatly improves robustness over fixed nonlinear models; however they do not consider uncertainty with the friction coefficient estimate, or other parameters in the Pacejka curve. For estimation of the full tire force curve, recent algorithms have modeled it as a Gaussian process (GP) with unknown and time-varying mean and covariance function [RW06], leading to a GP state-space model (GP-SSM). Due to the nonparametric nature of the GP, this method is not subject to specific modeling constraints that various tire models impose. Importantly the method is insensitive to overfitting to the data, and provides an estimate of uncertainty in explored and unexplored state space regions. [HKZ19] proposes using an MPC controller that uses such a Gaussian process to model the unknown parts of a vehicle dynamic model as an additive component. To more explicitly model tire forces, a recently proposed approach for real-time joint state estimation and learning of the tire force function [Ber21], combines a particle filter [DJ09] with a computationally efficient formulation of GP-SSMs for jointly estimating online the state and associated state-transition function, although this approach has yet to be integrated into a controller or motion planner.

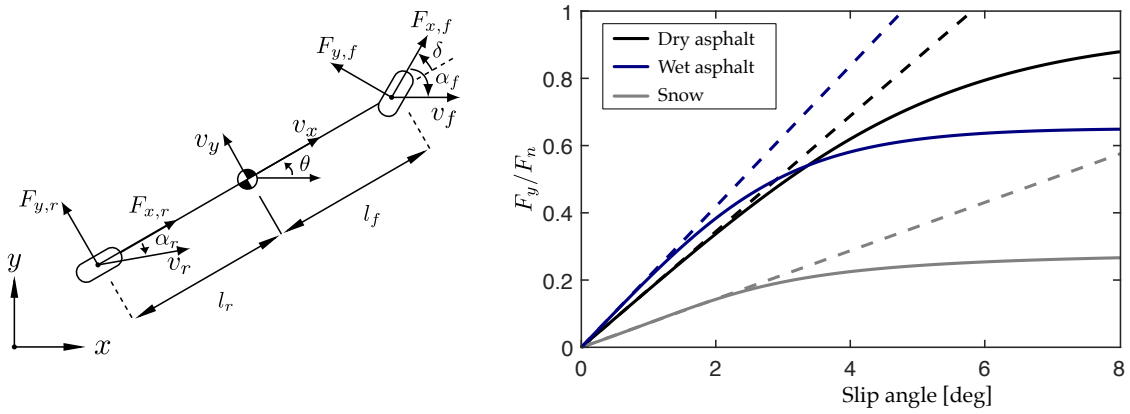
7.2 Dynamic Models

This section describes the dynamic models used in this chapter. §7.2.1 described the high-fidelity model used for simulation. §7.2.2 describes assumptions about measurements. §7.2.3 gives the general form of the planning model.

7.2.1 High-fidelity Model

Consider the single-track vehicle model, where the left and right track of the car are lumped into a single centered track, as Fig. 7.1 (a) shows. Hence, only a single front and a single rear tire are considered, and roll and pitch dynamics are ignored, resulting in two translational and one rotational degrees of freedom.

The state vector is $z_{\text{hi}} = [x, y, \theta, v_x, v_y, \dot{\theta}, \delta]$ where v_x is the longitudinal velocity, v_y is the lateral velocity, $\dot{\theta}$ is the yaw rate, and δ is the front wheel angle. The inputs to the vehicle model are $u = [\omega_f,\ \omega_r,\ \dot{\delta}]$, where $(\omega_f,\ \omega_r)$ the front and rear wheel speeds and $\dot{\delta}$ is the tire-wheel angle rate of change. The single-track model lumps together the left and right wheel on each axle, and roll and pitch dynamics are neglected. As shown in [BOLN14], a single-track model is sufficiently accurate where the tire forces reach the nonlinear region but the maneuvers are not aggressive enough to result in large roll angles. Thus, the model has two translational and one rotational



(a) (b)

Figure 7.1: Subfigure (a) shows the single track bicycle model (7.1) used as the high-fidelity model in Chapter 7. (x, y, θ) are the global position and yaw. (v_x, v_y) are the velocities in the body-fixed frame. l_f and l_r are the distances to the front and rear axels. δ is the steering wheel angle. α_f and α_r are the front and rear slip angles. F indicates the tire forces at each wheel. Subfigure (b) shows a plot of normalized lateral tire force given by a Pacejka model (7.4) for snow, wet asphalt, and dry asphalt. The dashed lines indicate linear approximations which are valid at low slip angles.

degrees of freedom. The model dynamics are

$$m(\dot{v}_x(t) - v_y(t)\dot{\theta}(t)) = F_{x,f}(\cdot)\cos(\delta(t)) + F_{x,r}(\cdot) - F_{x,f}(\cdot)\sin(\delta(t)), \tag{7.1a}$$

$$m(\dot{v}_y(t) + v_x(t)\dot{\theta}(t)) = F_{y,f}(\cdot)\cos(\delta(t)) + F_{y,r}(\cdot) + F_{x,f}(\cdot)\sin(\delta(t)), \tag{7.1b}$$

$$I\ddot{\theta}(t) = l_f F_{y,f}(\cdot)\cos(\delta(t)) - l_r F_{y,r}(\cdot) + l_f F_{x,f}(\cdot)\sin(\delta(t)), \tag{7.1c}$$

where $F_x : \mathbb{R} \to \mathbb{R}$, $F_y : \mathbb{R}^2 \to \mathbb{R}$ give the longitudinal/lateral tire forces, the subscripts f, r stand for front and rear, respectively, m is the vehicle mass, I is the vehicle inertia about the vertical axis, δ is the front-wheel steering angle, and l_f and l_r are the distance from the front and rear axles to the center of mass. The normal force $F_{n,i}$ resting on each front/rear wheel are approximated as

$$F_{n,f} = mg(l_r/l), \qquad F_{n,r} = mg(l_f/l), \tag{7.2}$$

where the wheel base is $l = l_f + l_r$. The slip angles α_i and slip ratios σ_i are defined as in [PH06, Raj11],

$$\alpha_i = -\arctan\left(\frac{v_{y,i}}{v_{x,i}}\right), \qquad \sigma_i = \frac{r_w\omega_i - v_{x,i}}{\max(r_w\omega_i, v_{x,i})}, \tag{7.3}$$

where $i \in \{f, r\}$ and r_w is the wheel radius, and $v_{x,i}$ and $v_{y,i}$ are the longitudinal and lateral wheel velocities for wheel i with respect to an inertial system, expressed in the coordinate system of the

wheel. The tire forces are computed with the Magic Formula model [PH06], and combined loading is based on the friction ellipse as follows:

$$F_{x,i} = \mu_{x,i} F_{n,i} \sin(D_{x,i} \arctan(B_{x,i}(1 - E_{x,i})\sigma_i + E_{x,i} \arctan(B_{x,i}\sigma_i))), \tag{7.4}$$

$$F_{y,i} = \eta_i \mu_{y,i} F_{n,i} \sin(D_{y,i} \arctan(B_{y,i}(1 - E_{y,i})\alpha_i + E_{y,i} \arctan(B_{y,i}\alpha_i))), \tag{7.5}$$

$$\eta_i = \sqrt{1 - \left(\frac{F_{x,i}}{\mu_{x,i} F_{n,i}}\right)^2}, \tag{7.6}$$

where $\mu_{j,i}$, $B_{j,i}$, $D_{j,i}$ and $E_{j,i}$, for $i \in \{f, r\}, j \in \{x, y\}$, are the friction coefficients and stiffness, shape, and curvature factors. Note that in (7.4), the longitudinal force does not explicitly depend on the lateral slip, and it is possible to use more accurate models to represent the combined slip [BOLN14, PH06].

7.2.2 Measurements

This chapter focuses on executing maneuvers with large changes in the lateral direction, hence the implementation is primarily concerned with estimating the lateral tire forces. In this implementation, the vehicle receives measurements of its pose $(\hat{x}, \hat{y}, \hat{\theta})$ from a localization algorithm. The vehicle has an inertial measurement unit, providing a measurement of the longitudinal and lateral accelerations $(\hat{a}_x, \hat{a}_y)$ and yaw rate $\hat{\dot{\theta}}$. Note that in (7.1) $a_y(t) = \dot{v}_y(t) + v_x(t)\dot{\theta}(t)$ and $a_x(t) = \dot{v}_x(t) - v_y(t)\dot{\theta}(t)$. The inputs $\omega_f,\ \omega_r$ and state δ are also measured with negligible uncertainty. In practice encoders can provide accurate measurements of these quantities. Two important assumptions are made for the implementation:

Assumption 81. *Noise of the vehicle measurements error, e, is normally distributed with zero mean with known co-variance, $e \sim \mathcal{N}(0, \Sigma_{\text{meas}})$. This is reasonable since the measurement noise can oftentimes be determined from prior experiments and data sheets, and offset to the mean can represented as a bias term that can also be estimated [Gus10].*

and

Assumption 82. *The vehicle's longitudinal velocity, v_x, is estimated from a combination of the wheel speed sensors and longitudinal acceleration. For examples of such estimators see [TSC06, GFX13].*

The estimation algorithms presented could estimate both the lateral and longitudinal forces and velocities as described in [BDC18].

7.2.3 Planning Model

The planning model that used for trajectory optimization in this chapter will take the form of the single track bicycle model (7.1), however the controller does not have access to (or use) the true tire force functions (7.4-7.6). Instead these functions are estimated given vehicle data. The planning model takes a form similar to (3.13)

$$\dot{z}(t) = f(t, z, u) + g(t, z, u)d(\cdot), \tag{7.7}$$

with a few differences. First the notation u is used instead of k, since the input form (wheel angle command and acceleration) are the same as the high-fidelity model. Second instead of allowing d to vary between $[-1, 1]$, as in Chapter 3, the probability of its value is estimated online. In §7.3.1, d is time-varying parameter vector that is estimated. In §7.3.2, d will consist of state-dependent basis functions, whose coefficients are estimated.

7.3 Tire Force Estimation

This section describes to models used in the trajectory optimization (MPC) in §7.4 and the algorithms used for estimating uncertainty in the tire forces. §7.3.1 describes a linear tire model and estimation technique [BDC18]. §7.3.2 describes a nonlinear tire model and estimation technique [BQV21].

7.3.1 Stiffness Estimator

The tire-stiffness estimator for a linear tire model is a recently developed adaptive particle-filter approach. The contribution in this dissertation is not the development of the algorithm, but the integration of it into the MPC formulation as described in §7.4.2. An extension of the algorithm to estimate the vehicle's mass and moment of inertia has also been developed and integrated into the MPC framework [BQV21]. A brief summary of the algorithm and details of the implementation are provided here, for a more extensive description of the algorithm see [BDC18]. This section focuses on lateral dynamics, so the lateral velocity and yaw rate $z^e = [v_y, \dot{\theta}]$ are estimated, with inputs $u^e = [v_x, \delta]$, and measurement vector $y^e = [\hat{a}_y, \hat{\dot{\theta}}]^\top$. An important feature of the estimator is that it only relies on sensors commonly available in production vehicles.

The method employs the single-track vehicle model (7.1) but uses and a linear approximation of the front and rear tire forces,

$$F_{x,i} \approx C_{x,i}\sigma_i, \qquad\qquad F_{y,i} \approx C_{y,i}\alpha_i, \tag{7.8}$$

where $C_{x,i}$ and $C_{y,i}$ are the longitudinal and lateral stiffness, respectively. As seen in Figure 7.1 (b), the linear approximation is valid at low slip values. Because of this, in the implementation, the estimator is activated only when the wheel angle and slip ratios are within a predefined threshold. Additionally, the estimator is deactivated when the wheel angle is near zero since the system becomes unobservable [BDC18]. Since the estimator is focused on estimating the lateral cornering stiffness and I assumed $C_{x,i} \approx 2C_{y,i}$. This approximation provides a coarse update of the longitudinal stiffness. The linear relationship was chosen based on the models used in simulation and could alternatively be fit with experimental data. Since the maneuvers in this work will not require large longitudinal accelerations, accurately modelling the longitudinal stiffness is not critical to the controller performance.

The stiffness values in (7.8) are decomposed into a nominal and unknown part,

$$\begin{bmatrix} C_{y,f} \\ C_{y,r} \end{bmatrix} = \begin{bmatrix} C_{y,f}^{\text{nom}} \\ C_{y,r}^{\text{nom}} \end{bmatrix} + \xi, \tag{7.9}$$

where $C_{y,i}^{\text{nom}}$ is the nominal value of the cornering stiffness, for example, a priori determined on a nominal surface, and $\xi \in \Xi \subset \mathbb{R}^2$ is a time-varying, unknown part. The unknown stiffness components are modeled as random process noise acting on the otherwise deterministic system.

Assumption 83. *The realization of ξ_j at time $t = t_j$ is normally distributed according to $\xi_j \sim \mathcal{N}(\hat{C}_j, \Sigma_j)$, where $\hat{C}_j$ and Σ_j are the unknown, time-varying, mean and covariance.*

Inserting (7.8)–(7.9) into (7.1) and discretizing using forward-Euler with a sampling period τ_s gives the discrete-time dynamics

$$z_{j+1}^e = z_j^e + \tau_s f^e(z_j, u_j^e) + \tau_s g^e(z_j, u_j^e)\xi_j, \tag{7.10}$$

$$f(z^e, u^e) = \begin{bmatrix} \frac{1}{m}\left(C_{y,f}^{\text{nom}}(2\sigma_f \sin\delta + \alpha_f \cos\delta) + C_{y,r}^{\text{nom}}\alpha_r\right) - v_x\dot{\theta} \\ \frac{1}{I}\left(C_{y,f}^{\text{nom}} l_f(2\sigma_f \sin\delta + \alpha_f \cos\delta) - C_{y,r}^{\text{nom}} l_r \alpha_r\right) \end{bmatrix}, \tag{7.11}$$

$$g(z^e, u^e) = \begin{bmatrix} \frac{1}{m}(2\sigma_f \sin\delta + \alpha_f \cos\delta) & \frac{1}{m}\alpha_r \\ \frac{l_f}{I}(2\sigma_f \sin\delta + \alpha_f \cos\delta) & \frac{-l_r}{I}\alpha_r \end{bmatrix}, \tag{7.12}$$

where the subscript j refers to the current timestep. The estimator uses the lateral acceleration and yaw-rate measurements and models the bias b_j of the inertial measurements as a random walk;

which, again by inserting (7.8)–(7.9) into (7.1), can be writen in the form

$$y^e = h(z_j^e, u_j^e) + b_j + d(z_j^e, u_j^e)\xi_j + e_j, \tag{7.13}$$

$$h(z^e, u^e) = \begin{bmatrix} \frac{1}{m}\left(C_{y,f}^{\text{nom}}(2\sigma_f \sin\delta + \alpha_f \cos\delta) + C_{y,r}^{\text{nom}}\alpha_r\right) \\ \dot{\theta} \end{bmatrix}, \tag{7.14}$$

$$d(z^e, u^e) = \begin{bmatrix} \frac{1}{m}(2\sigma_f \sin\delta + \alpha_f \cos\delta) & \frac{1}{m}\alpha_r \\ 0 & 0 \end{bmatrix}, \tag{7.15}$$

were e is the zero-mean Guassian measurement noise as in Assumption 81. The particle filter from [BDC18] jointly estimates the state vector and process noise. It has n particles represents the probability densities of the state as in [BDC18, (42)]

$$p(z_j^e \mid y_{0:j}^e) \approx \sum_{i=1}^{n} q_j^{(i)} \delta_0(z_j^e - z_j^{e,(i)}) \tag{7.16}$$

where δ_0 is the dirac delta and $z_j^{e,(i)}$ is the state estimate for the i^{th} particle at timestep j. Each particle of the estimator contains, an estimated of the noise, ξ_j, which is modeled as a Normal-inverse-Wishart distribution defined by statistics $\gamma, \hat{C}, \Lambda, \nu$ as in [BDC18, (25)]:

$$p(\xi_j \mid z_{0:j}^e, y_{0:j}^e) = \text{NiW}(\gamma_{j|j}, \hat{C}_{j|j}, \Lambda_{j|j}, \nu_{j|j}). \tag{7.17}$$

To extract values for the mean and co-variance a weighted average over the particles is used, as in [BDC18, (27)]:

$$\hat{C}_j \approx \sum_{i=1}^{n} q_j^{(i)} \hat{C}_{j|j}^{(i)} \tag{7.18}$$

$$\Sigma_j \approx \sum_{i=1}^{n} q_j^{(i)} \left(\frac{1}{\nu_{j|j} - 4} \Lambda_{j|j}^{(i)} + (\hat{C}_{j|j}^{(i)} - \hat{C}_j)(\hat{C}_{j|j}^{(i)} - \hat{C}_j)^\top \right) \tag{7.19}$$

Figure 7.2 shows the output from the stiffness estimator for the vehicle described by (7.1) on a surface switching from dry asphalt to snow and back. The estimator uses a sampling period of $\tau_s = 0.01$ s. The "true stiffness" is defined as the slope of the tire-force curve at $\alpha = 0$. In Figure 7.2, the true stiffness is underestimated at times on both surfaces, as a result of tire saturation. Figure 7.3 provides a simple illustration of why this occurs for an asphalt tire model. When the vehicle is operating at nonzero slip angles, the estimated tire-force model can be thought of as a line between the origin and the true tire force. The slope of this line (the estimated stiffness) decreases as the slip angle increases and the tire-force curve flattens. In §7.4.2 this is leveraged to

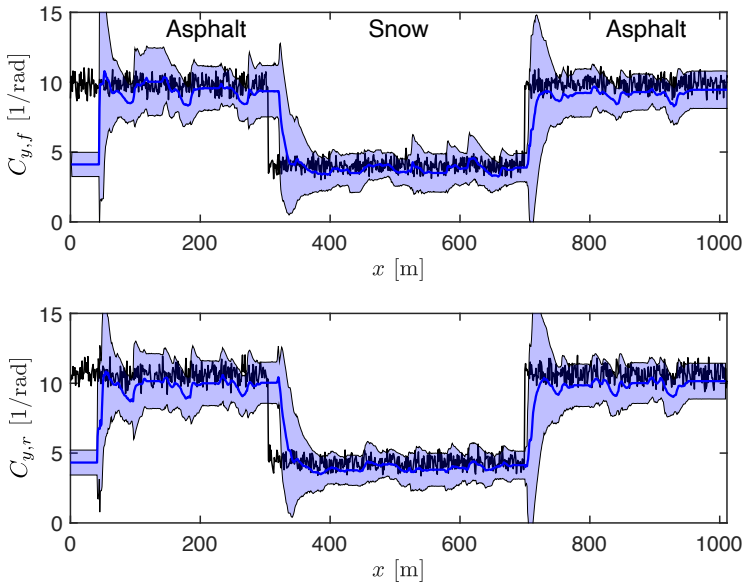


Figure 7.2: Stiffness estimates for a surface switching from dry asphalt to snow and back. The black line is the true stiffness, the slope of the nonlinear tire-force curve at $\alpha = 0$. The blue solid and shaded regions are the mean and 95% confidence interval from the stiffness estimator. The true stiffness is underestimated when the tires saturate.

develop constraints that maintain stability even when the vehicle is near the nonlinear regions of the tire curve.

7.3.2 Nonlinear Estimator

This section briefly describes the Bayesian tire-friction estimator that approximates the friction function F entering the vehicle dynamics model (7.1) as a Gaussian process (GP) with unknown mean and covariance function. The estimator is targeted for embedded automotive-grade hardware and sensors. I used a recently developed method for jointly estimating the tire-force function and the vehicle state only using sensors available in production cars, namely wheel-speed sensors and inexpensive accelerometers and gyroscopes. This section focuses on estimating the lateral tire forces, so $F = [F_{y,f}\ F_{y,r}]^\top$, but the algorithm can be extended to estimate longitudinal forces as well. I briefly outline the formulation of the method here and refer the reader to [Ber20, Ber21] for a more complete description.

7.3.2.1 Estimation Model

In this method the state and input vector for the estimator are defined as $z^e = [v_y, \dot{\theta}]^\top$, $u^e = [\delta, v_x]^\top$. For brevity, define the vector $\alpha = [\alpha_f, \alpha_r]^\top$ and model the lateral tire force function

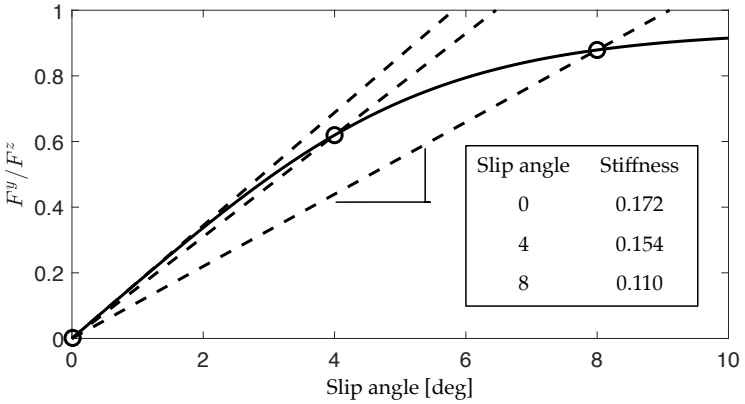


Figure 7.3: Linear approximation of the tire forces at slip angles of 0, 4, 8 deg for an asphalt tire model. The solid line is the true tire-force curve, while dashed lines are the linear approximations. The stiffness (slope of the line) decreases as the slip angle increases.

$F = [F_{y,f}\ F_{y,r}]^\top$ as a realization from a GP with mean function $\hat{F}$ and covariance function Σ,

$$F(\alpha(z^e, u^e)) \sim \mathcal{GP}(\hat{F}(\alpha(z^e, u^e)), \Sigma(\alpha(z^e, u^e))). \tag{7.20}$$

The resulting vehicle SSM is a GP-SSM where the tire friction is a GP. A bottleneck in some of the proposed GP-SSM methods is the computational load. In this section, I use a computationally efficient reduced-rank GP-SSM framework, where the GP is approximated as a basis function expansion using the Laplace operator eigenvalues and eigenfunctions

$$\phi_k(\alpha) = \frac{1}{\sqrt{L}} \sin\left(\frac{\pi k(\alpha + L)}{2L}\right), \quad \lambda_k = \left(\frac{\pi k}{2L}\right)^2, \tag{7.21}$$

defined on the interval $[-L, L]$, such that

$$F_i \approx \sum_{k=1}^{m} \xi_{i,k}\, \phi_k(\alpha_i), \tag{7.22}$$

where $i \in \{f, r\}$ and the weights $\xi_{i,k}$ are Gaussian random variables with unknown mean and covariance, whose prior depends on the spectral density that is a function of the eigenvalues in

(7.21). The basis-function expansion can be written in matrix form as

$$F = \underbrace{\begin{bmatrix} \xi_{f,1} & \cdots & \xi_{f,m} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \xi_{r,1} & \cdots & \xi_{r,m} \end{bmatrix}}_{\Gamma = \begin{bmatrix} \Gamma_f & 0 \\ 0 & \Gamma_r \end{bmatrix}} \underbrace{\begin{bmatrix} \phi_1(\alpha_f) \\ \vdots \\ \phi_m(\alpha_f) \\ \phi_1(\alpha_r) \\ \vdots \\ \phi_m(\alpha_r) \end{bmatrix}}_{\varphi(\alpha(z^e, u^e))}, \tag{7.23}$$

where $\xi_{i,k}$ are the weights to be learned and m is the total number of basis functions. For convenience, the prior on the coefficients at time step $j = 0$ is a zero-mean matrix-normal ($\mathcal{MN}$) distribution. This requires writing out tire force estimate as a deviation from a nominal model, similar to (7.9). With the matrix form (7.23), the vehicle dynamics model used in the estimator can be written as

$$z^e_{j+1} = f^e(z^e_j, u^e_j) + g^e(z^e_j, u^e_j)\Gamma\varphi(\alpha(z^e_j, u^e_j)), \tag{7.24}$$

where j indicates the current timestep. Hence, the original problem of learning the infinite-dimensional friction function F has been transformed to learning the matrix Γ in (7.24), which is substantially easier to do in an online setting. The lateral velocity and yaw rate are estimated as part of the state according to (7.1). The measurement model can also be written as

$$y^e_j = h(z^e_j, u^e_j) + d(z^e_j, u^e_j)\Gamma\varphi(\alpha_j(z^e_j, u^e_j)) + e_j, \tag{7.25}$$

where the measurement noise e is as in Assumption 81.

Remark 84. *According to well-established tire models (e.g., [PH06], see Fig. 7.1 (b)), it is known that the tire-friction function is antisymmetric. At the same time, the basis-function expansion* (7.22) *is a weighted sum of sinusoids for each dimension. Hence, antisymmetry can be enforced by restricting the sum in* (7.22) *to even values for k (i.e., $k = 2, 4, \ldots, m$). This substantially reduces the number of parameters to estimate and at the same time ensures that the estimated tire force function. F, passes through the origin.*

7.3.2.2 Joint State and Friction-Function Learning

The vehicle state needs to be estimated concurrently with the friction function, which will be easier to estimate reliably for moderate slip angles corresponding to normal driving, and the measurements are automotive grade and therefore have significant noise. Hence, to properly account

for the inherent uncertainty the estimation problem is approached in a Bayesian framework. Due to the nonlinearities of the tire-friction function and the vehicle model, the estimation problem is highly non-Gaussian. I therefore use a particle-filter based estimator that estimates the vehicle state z^e and basis-function weights Γ [Ber20]. The particle filter approximates the joint posterior density

$$p(\Gamma_j, z^e_{0:j}|y^e_{0:j}) \tag{7.26}$$

at each time step $t = t_j$, that is, the posterior density function of the matrix Γ_j of basis-function weights and state trajectory $z^e_{0:j} = \{z^e_0, \dots, z^e_j\}$ from time step 0 to j, given the measurement history $y^e_{0:j} = \{y^e_0, \dots, y^e_j\}$. I use the standard decomposition of (7.26) into

$$p(\Gamma_j, z^e_{0:j}|y^e_{0:j}) = p(\Gamma_j|z^e_{0:j}, y^e_{0:j})\, p(z^e_{0:j}|y^e_{0:j}). \tag{7.27}$$

The two densities on the right-hand side of (7.27) can be estimated recursively. The state trajectory density $p(z^e_{0:j}|y^e_{0:j})$ is estimated by a set of n weighted state trajectories as

$$p(z^e_{0:j}|y^e_{0:j}) \approx \sum_{i=1}^{n} q_j^{(i)}\, \delta_0(z^e_{0:j} - z^{e,(i)}_{0:j}), \tag{7.28}$$

where $q_j^{(i)}$ is the weight of the i^{th} state trajectory $z^{e,(i)}_{0:j}$ and $\delta_0(\cdot)$ is the Dirac delta. Given the state trajectory, the sufficient statistics necessary to approximate $p(\Gamma_j|z^{e,(i)}_{0:j}, y^e_{0:j})$ can be computed for each particle. Because the state trajectory is determined from the particle filter, the computations leading up to the estimation of $p(\Gamma_j|z^{e,(i)}_{0:j}, y^e_{0:j})$ are analytic.

To determine the covariance and mean function, the state trajectory can be marginalized from $p(\Gamma_j|z^{e,(i)}_{0:j}, y^e_{0:j})$ according to

$$p(\Gamma_j|y^e_{0:j}) = \int p(\Gamma_j|z^{e,(i)}_{0:j}, y^e_{0:j}) p(z^{e,(i)}_{0:j}|y^e_{0:j})\, dz^e_{0:j} \approx \sum_{i=1}^{n} q_j^{(i)}\, p(\Gamma_j^{(i)}|z^{e,(i)}_{0:j}, y^e_{0:j}), \tag{7.29}$$

and each particle retains its own estimate of Γ_j together with the weight $q_j^{(i)}$. Then, the mean and covariance can be estimated by choosing the i^{th} particle that fulfills $i^* = \arg\max_{i\in\{1,\dots,n\}} q_j^{(i)}$. This results in the following expressions for the friction estimate and associated covariance function at

timestep j:

$$\hat{F}_j(\alpha(z^e, u^e)) = \begin{bmatrix} \hat{\Gamma}_{f,j}^{(i^*)} \varphi(\alpha_f(z^e, u^e)) \\ \hat{\Gamma}_{r,j}^{(i^*)} \varphi(\alpha_r(z^e, u^e)) \end{bmatrix}, \tag{7.30a}$$

$$\Sigma_j(\alpha(z^e, u^e)) = \begin{bmatrix} \varphi(\alpha_f(z^e, u^e))^\top V_{f,j}^{(i^*)} \varphi(\alpha_f(z^e, u^e)) Q_f & 0 \\ 0 & \varphi(\alpha_r(z^e, u^e))^\top V_{r,j}^{(i^*)} \varphi(\alpha_r(z^e, u^e)) Q_r \end{bmatrix}, \tag{7.30b}$$

where the matrix $\hat{\Gamma}_{k,j}^{(i)}$ is the particle's mean estimate and $V_{k,j}^{(i)}$ and Q_k, for $k \in \{f, r\}$, are parts of the sufficient statistics when determining (7.29)—more details can be found in [Ber20]. Empirically I found that there is little difference between using the maximum likelihood particle and averaging over all particles, hence I chose to use the most likely particle for computational efficiency in the MPC integration §7.4.3.

7.3.2.3 Implementation Results

Fig. 7.4 shows the front tire-force estimates in closed loop for a sequence of lane change manuevers with a surface change from asphalt to snow to asphalt. 500 particles and 10 basis functions were used. The estimates are initialized for the parameters on snow. At 9.0 s the vehicle transitions to snow, and the learned model has fully learned about the surface change by 11.0 s. The snow transition takes place during a portion of the trajectory where the observability of the lateral dynamics is poor. The vehicle transitions back to asphalt at 16.25 s and the learned model (at 18.5 s) is slower to converge to the asphalt model compared to the snow transition. This is because it has not fully explored the tire force curve on asphalt yet. For the region of the slip angle corresponding to where data has been acquired, the estimates follow closely to the true friction model, and it is clear that where small amounts of data have been gathered, for example, for slip angles close to or beyond the peak, the uncertainty increases accordingly.

7.4 Adaptive, Chance-constrained Nonlinear Model Predictive Control

This section describes the chance-constrained Nonlinear Model Predictive Control (NMPC) Algorithm that, given the current tire force estimate, is solved to generate a safe trajectory and control inputs for the vehicle. The algorithm is executed in a receding-horizon framework, similar to Algorithm 1. §7.4.1 summarizes the problem formulation and solver used. The following two sections §7.3.1 and §7.4.3 explain the implementations with the linear and nonlinear estimators from §7.3.

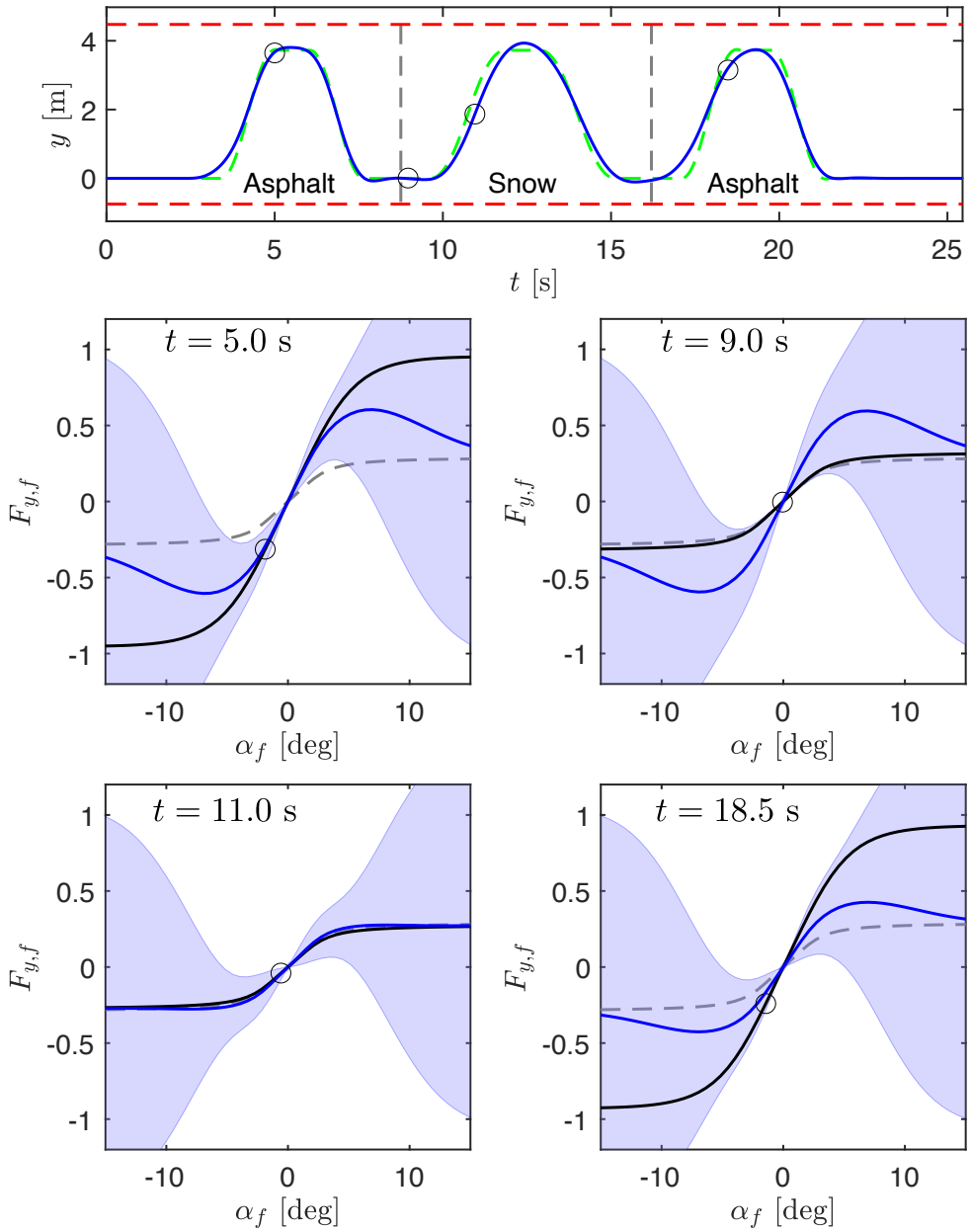


Figure 7.4: The top plot shows the closed-loop trajectory (blue) of a (7.1) tracking lane change maneuvers (green) on asphalt, snow then asphalt. The bottom plots show the front tire force at $t =$ 5,9,11,and 18.5 s. The black circles correspond to the vehicle state at each timestep. The black line is the simulation model. The blue line is the mean function $\hat{F}_{y,f}$ (7.30a). The blue shaded region is the 95% confidence interval from (7.30b). The gray dashed line is the nominal tire force model (snow). At 9.0 s the vehicle transitions to snow, and the learned model has fully learned about the surface change by 11.0 s. The vehicle transitions back to asphalt at 16.25 s and the learned model is slower to converge to the asphalt model.

7.4.1 Problem Formulation and Solver

This section summarizes the chance-constrained trajectory optimization program. The program is

$$\begin{aligned} \min_{z(\cdot),u(\cdot)} \ & \mathbb{E}\left[\int_0^T J(z(t),u(t))dt\right] \\ \text{s.t. } & \dot{z}(t) = f(z(t),u(t),\xi), \\ & \Pr\left(\bigvee_{t\in[0,T],i\in\{1,\ldots,n_{\text{cons}}\}} h_i(z(t),u(t)) \geq 0\right) \leq \Delta \end{aligned} \tag{7.31}$$

where $z : [0,T] \to Z$ and $u : [0,T] \to U$ are functions describing the state trajectory and inputs, $f : Z \times U \times \Xi \to \mathbb{R}^{n_Z}$ is a dynamic model of the system. Note that in this application, the planning models are of the form (7.7); however here and in §7.4.1.1, f is written as a general function. Each $h_i : Z \times U \to \mathbb{R}$ is a constraint, whose probability of violation is limited, and $J : Z \times U \to \mathbb{R}$ is the cost function, whose expectation is minimized. In this work the cost function considered is

$$J(z,u) = \frac{1}{2}\|z - z_{\text{ref}}\|_Q^2 + \frac{1}{2}\|u - u_{\text{ref}}\|_R^2. \tag{7.32}$$

where $z_{\text{ref}} : [0,T] \to Z$ and $u_{\text{ref}} : [0,T] \to U$ specify a desired reference trajectory and $Q \in \mathbb{R}^{n_Z \times n_Z}$ and $R \in \mathbb{R}^{n_Z \times n_Z}$ are positive semidefinite and definite matrices. In this context, the system dynamics, f, are given by (7.1), where the functions defining the tire forces are approximated by the tire force estimators in §7.3.

7.4.1.1 Discrete time NMPC Formulation

The formulation in [FDCQ20], solves a discrete time approximation of (7.31). The time horizon is partitioned into a set of n_{pred} discrete time steps $t_j \in [0,T]$, such that $t_j = \frac{j}{n_{\text{pred}}}T$. The inputs are parameterized in the form of a feedback controller with an offset changing at each discrete timestep

$$\tilde{u}_j = u_{\text{ref},j} + \mathcal{B}(z_j - z_{\text{ref},j}) + u_j, \tag{7.33}$$

where $\mathcal{B} \in \mathbb{R}^{n_Z \times n_U}$ is a matrix of feedback gains. The formulation considers a discrete-time system of the dynamics

$$z_{j+1} = f_1(z_j, \tilde{u}_j, \xi_j), \tag{7.34}$$

where the subscript j indicates the state, input, and uncertainty vector estimate at time t_j and f_1 numerically approximates the integration of the system dynamics forward by 1 timestep. In this

work rk4 integration is used [PWT+07, §17.1]. In this work $\mathcal{B}$ is computed by solving the Ricatti equation for a linearized version of the discrete time system [SP08]. At the k^{th} planning iteration, based on the expected state estimate $\hat{z}_k$ and covariance S_k, the Chance-constrained NMPC solves

$$\min_{z,u,S} \sum_{j=0}^{n_{\text{pred}}} J(z_j, \tilde{u}_j) \tag{7.35}$$

$$\text{s.t.} \begin{cases} \forall j \in \{0, \dots, n_{\text{pred}} - 1\}, i \in \{1, \dots, n_{\text{cons}}\} \\ 0 = z_{j+1} - f_1(z_j, \tilde{u}_j, \hat{\xi}_k),\ z_0 = \hat{z}_k, \\ S_{j+1} = A_j S_j A_j^\top + B_j \Sigma_k B_j^\top,\ S_0 = S_k, \\ \Pr\left(h_i(z_j, \tilde{u}_j) \geq 0\right) \leq \Delta, \end{cases}$$

where the Jacobian matrices read as $A_j = \frac{\partial f}{\partial z}(z_j, \tilde{u}_j, \hat{\xi}_k)$ and $B_j = \frac{\partial f}{\partial \xi}(z_j, \tilde{u}_j, \hat{\xi}_k)$, and $\hat{\xi}_k$ and Σ_k signify the mean and covariance of the uncertainty. The state covariance propagation equations correspond to the extended Kalman filtering (EKF) approach, similar to [TVC+15b]. In addition to the dynamics and covariance propagation, there are a few other approximations with respect to how the uncertainty vector ξ is treated in (7.35) compared to (7.31). First the cost function is evaluated with respect to the expected (mean) values of the initial condition and uncertainty vector. Second, the marginal probability of each chance constraint is enforced, meaning conditional probability on constraint violations is not accounted for. One could account for this by reducing the probability of constraint violation to satisfy Boole's inequality as in (6.47), [BLW06], or considering an iterative risk allocation strategy as proposed in [OW08]; although the later comes at increase computational cost since an additional risk allocation problem must be solved at every planning iteration.

7.4.1.2 Chance Constraint Approximation

The probabilistic chance constraints in (7.35) are approximated with deterministic constraints as in [TVC+15b], where the i^{th} constraint is written as

$$h_i(z_j, \tilde{u}_j) + \eta \sqrt{\frac{\partial h_i}{\partial z_j} S_j \frac{\partial h_i}{\partial z_j}^T} \leq 0, \tag{7.36}$$

where η is referred to as the back-off coefficient and depends on the desired probability threshold Δ and assumptions about the resulting state distribution. The backoff coefficient for Cantelli's inequality, $\eta = \sqrt{\frac{1-\Delta}{\Delta}}$, holds regardless of the underlying distribution but is conservative. In this work, I assumed normally-distributed state trajectories and set

$$\eta = \sqrt{2}\text{erf}^{-1}(1 - 2\Delta), \tag{7.37}$$

where $\text{erf}^{-1}(\cdot)$ is the inverse error function.

7.4.1.3 Implementation

The MPC program is implemented on the vehicle, described in §7.2 in simulation as in Algorithm 13

Algorithm 13 Adaptive Chance-constrained MPC Simulation

1: **Require:** $z_{\text{hi},0}$, Σ_0, $\hat{z}_0$, S_0, z_{ref}, u_{ref}, cost matrices $Q,\ R$, MPC planning time $\tau_{\text{plan}} > 0$
2: **Initialize:** $k = 0$, $\hat{\xi}_0 = 0$
3: **Loop:** //Lines 6 - 8 run concurrently
4: $(z^*, u^*, S^*) \leftarrow$ `solveChanceConstrainedMPC`$(\hat{z}_k, S_k, \hat{\xi}_k, \Sigma_k, z_{\text{ref}}, u_{\text{ref}}, Q, R)$
5: Compute input $\tilde{u}_k$ from (7.33) with (z_0^*, u_0^*)
6: $z_{\text{hi},k+1} \leftarrow$ `simulateModel`$((k+1)\tau_{\text{plan}}, z_{\text{hi},k}, \tilde{u}_k)$
7: $\{y^e\} \leftarrow$ `getMeasurements`$(\{z_{\text{hi}}(t) | t \in [k\tau_{\text{plan}}, (k+1)\tau_{\text{plan}}]\})$
8: $(\hat{z}_{k+1}, P_{k+1}, \hat{\xi}_{k+1}, \Sigma_{k+1}) \leftarrow$ `updateStateEstimate`$(\{y^e\})$
9: **End**

Algorithm 13 begins with initial estimates of mean and covariance for the vehicle state an uncertainty vector. It first solves (7.35) and takes the first input $(j = 0)$ from the solution, and applies it to the simulation model (7.1) which uses the nonlinear Pacejka model (7.4). In this step, one can introduce random perturbations into the Pacejka parameters during the simulation. The measurements as described in §7.2 are given to the estimator, which updates the mean and covariance of the state and uncertainty vector. In implementations, the estimators are usually run at a higher frequency than the MPC program, so I specify that this occurs while the simulation model is integrated.

To solve (7.35), I used the SNMPC implementation in [FDCQ20], based on an SQP optimization algorithm in which a series of QP approximations are solved using the PRESAS QP solver [QDC20]. The nonlinear function and derivative evaluations, for the preparation of each SQP subproblem, are performed using algorithmic differentiation (AD) and C code generation in CasADi [AGH+18a]. The algorithm uses a tailored Jacobian approximation along with an adjoint-based SQP method that allows for the numerical elimination of the covariance matrices from the SQP subproblem, which reduces the computation time when compared to standard SQP formulations for SNMPC [FDCQ20]. Note that one SQP iteration per control time step is typically performed for real-time implementations of NMPC, as discussed in [GZQ+20b]. Slack variables in the program formulation ensure that a feasible solution is always found in Algorithm 13 Line 4 (see Appendix C).

7.4.2 Formulation with Stiffness Estimator

To integrate the stiffness estimator in §7.3.1 with the NMPC program in §7.4.1.1, I used the single track model (7.1) with linear tire force functions (7.8) as the dynamics, f. The mean and covariance of the cornering stiffnesses are updated with the current estimate, i.e. in Algorithm 13 Line 8 $\hat{\xi}_k$ and Σ_k are updated with (7.18) ad (7.19). Note that by (7.10), the planning dynamics can take the form of (7.7).

7.4.2.1 Constraints

The following inequality constraints are enforced in the optimal-control problem of (7.35):

$$y_{\min} \leq y \leq y_{\max}, \tag{7.38a}$$

$$|\delta| \leq \delta_{\max}, \quad |\dot{\delta}| \leq \dot{\delta}_{\max}, \tag{7.38b}$$

$$|\sigma_i| \leq \sigma_{\max},\ i \in \{f, r\}, \tag{7.38c}$$

$$|\dot{\theta} v_x| \leq 0.85\hat{\mu} g, \quad \left|\frac{v_y}{v_x}\right| \leq \tan^{-1}(0.02\hat{\mu} g). \tag{7.38d}$$

Eq. (7.38a) bounds the lateral position, and is used to ensure that the vehicle stays on the road. Obstacle avoidance constraints could be considered in future work. Eqs. (7.38b)-(7.38c) bound the wheel angle, wheel angle rate, and slip ratios. The constraints in (7.38d) prevent the vehicle from entering regions of high lateral acceleration and side slip, and can be found in [Raj11, Chapter 8]. These equations in (7.38d) are refered to as *stability constraints*.

The stability constraints depend on the road friction μ, a parameter whose estimation is widely studied [KET17]. In this work, I estimated the friction based on the cornering stiffness. Experimental studies suggest that using a monotonic relationship is sufficient to differentiate between asphalt and snow [Gus97, APT12]. I used a linear relationship to approximate the road friction as a function of the cornering stiffness estimate,

$$\hat{\mu} \approx \min\left((a/2)(C_{y,f}^{\text{nom}} + \hat{C}_{y,f} + C_{y,r}^{\text{nom}} + \hat{C}_{y,r}), 1\right), \tag{7.39}$$

where a is a constant that was fit from the Pacejka models for asphalt and snow. This relationship proved to be effective in the simulations; finding an optimal relationship to use could be the subject of future work. The central idea of (7.39) is that the bounds on the acceleration and sideslip should tighten as the road friction, and consequently the cornering stiffness, decreases. For surfaces such as wet asphalt, which may have a high cornering stiffness but lower road friction, (7.39) is conservative in practice because the stiffness estimator underestimates the true stiffness as the tires saturate (as in Figure 7.3). Note that in the implementations slack variables to ensure feasibility

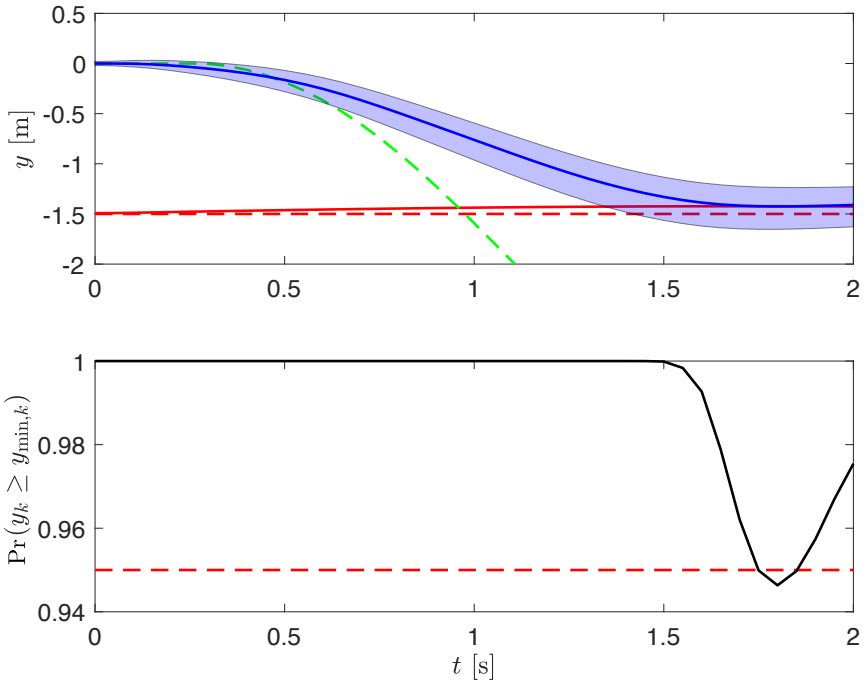


Figure 7.5: Illustration of a chance constraint enforced on the lateral position. In the top plot, the green dashed line is the reference trajectory. The red dashed line is the lateral constraint, and the red solid line is the tightened, chance constraint. The blue line is the mean trajectory from solving (7.35). The light blue shaded region indicates the upper and lower bounds for 1e5 random disturbance realizations. The lower plot shows the probability of constraint satisfaction of the 1e5 simulations (black) and the desired threshold (red). The chance constraint for $\Delta = 0.05$ is approximated to within 1%

(see Appendix C).

7.4.2.2 Illustrative Example for SNMPC Formulation

A control reference z_{ref} is set up that intentionally violates the lower constraint on the lateral position, to illustrate how the chance constraint is respected when (7.35) is solved. The solver is run for 30 SQP iterations. The mean cornering stiffness values, $\hat{C}$, correspond to a snow surface. The standard deviation is roughly 10% of the mean values. The least-squares cost (7.32) prioritizes the lateral position and wheel speed inputs. A timestep of $\tau_{\text{plan}} = 0.05$ s with a prediction horizon of $T = 2$ s is used. The solution trajectory is shown in Figure 7.5. The dynamic model is integrated forward for 1e5 disturbance realizations, and see that the chance constraint for $\Delta = 0.05$ is approximated to within 1%.

7.4.3 Formulation with Nonlinear Estimator

Using the single-track model (7.1) and the basis-function expansion of the lateral tire forced, $F_y \approx \Gamma\varphi(\alpha(z))$, the vehicle model (7.34) used in the NMPC program at timestep j can be written in the same format as (7.24)

$$z_{j+1} = f(z_j, \tilde{u}_j) + g(z_j, \tilde{u}_j)\Gamma\varphi(\alpha(z)). \tag{7.40}$$

The notation in this section differs from that in §7.3.2, where the slip angle is written as a function of the estimator states and inputs, since the wheel angle and longitudinal velocity are states in the MPC model.

7.4.3.1 Uncertainty Propagation

In the NMPC formulation [FDCQ20], the parametric uncertainty is modeled as a state-independent Gaussian random variable, $\xi \sim \mathcal{N}(\hat{\xi}, \Sigma)$. This leads to Jacobian matrices $A_j = \frac{\partial f}{\partial z}(z_j, \tilde{u}_j, \hat{\xi}_k)$ and $B_j = \frac{\partial f}{\partial \xi}(z_j, \tilde{u}_j, \hat{\xi}_k)$ as in (7.35). However, in this case, the friction uncertainty is a function that is modeled according to a GP at each time step k, $F_k(\alpha(z)) \sim \mathcal{GP}\left(\hat{F}_k(\alpha(z)), \Sigma(\alpha(z))\right)$. Hence, determining an equivalent formulation for the state-dependent uncertainty, by finding expressions for the involved Jacobians A_j, B_j is necessary. Starting from (7.40), by using the chain rule and $F_k \approx \Gamma_k\varphi(\alpha(z))$,

$$\begin{aligned} A_j &= \frac{\partial f}{\partial z}(z_j, \tilde{u}_j) + \frac{\partial g}{\partial z}(z_j, \tilde{u}_j)\hat{F}_k(\alpha(z_j)) + g(z_j, \tilde{u}_j)\frac{\partial \hat{F}_k(\alpha(z_j))}{\partial z} \\ &= \frac{\partial f}{\partial z}(z_j, \tilde{u}_j) + \frac{\partial g}{\partial z}(z_j, \tilde{u}_j)\hat{\Gamma}_k\varphi(\alpha(z_j)) + g(z_j, \tilde{u}_j)\hat{\Gamma}_k\frac{\partial \varphi}{\partial \alpha}(\alpha(z_j))\frac{\partial \alpha}{\partial z}(z_j), \end{aligned} \tag{7.41}$$

where $\hat{\Gamma}_k$ is the mean uncertainty estimate as in (7.30a). Here for ease of notation write $g_j = g(z_j, \tilde{u}_j)$ and $\varphi_j = \varphi(\alpha(z_j))$. The covariance propagation can be approximated as

$$\begin{aligned} \mathbb{E}[z_{j+1}z_{j+1}^\top] &\approx \mathbb{E}[(A_j z_j + g_j\hat{\Gamma}_k\varphi_j)(A_j z_j + g_j\hat{\Gamma}_l\varphi_j)^\top] \\ &= A_j S_j A_j^\top + \mathbb{E}[g_j\hat{\Gamma}_k\varphi_j\varphi_j^\top\hat{\Gamma}_k^\top g(z_j, \tilde{u}_j)^\top] \\ &= A_j S_j A_j^\top + \underbrace{g_j}_{B_j}\underbrace{\mathrm{cov}(\hat{\Gamma}_k\varphi_j)}_{\Sigma_k}g_j^\top, \end{aligned} \tag{7.42}$$

where $\mathtt{cov}(\cdot)$ corresponds to the covariance estimate in (7.30b). Eq. (7.42) allows one to use the NMPC formulation (7.35).

7.4.3.2 Constraints

The constraints in (7.35) are as follows

$$y_{\min} \leq y \leq y_{\max}, \tag{7.43a}$$
$$|\delta| \leq \delta_{\max}, \quad |\dot{\delta}| \leq \dot{\delta}_{\max}, \tag{7.43b}$$
$$\omega_{i,\min} \leq \omega_i \leq \omega_{i,\max},\ i \in \{f, r\}, \tag{7.43c}$$
$$\alpha_{i,\min} \leq \alpha_i \leq \alpha_{i,\max},\ i \in \{f, r\}. \tag{7.43d}$$

These are similar to the constraints in §7.4.2.1, however instead of constraining the slip ratio and adding the stability constraints, wheel speeds (7.43c) and slip angles (7.43d) are limited I found that this was sufficient, as this approach uses a nonlinear tire model, hence behaves more appropriately as the tires saturate. Note that in the presented implementation slack variables are incorporated to ensure feasibility (see Appendix C).

7.5 Results

This section offers a comparison of the proposed adaptive, chance-constrained Model Predictive Control (MPC) algorithms with determinsitic and non-adaptive MPC algorithms. §7.5.1 gives results for the implementation with the stiffness estimator described in §7.4.2. §7.5.2 gives results for the implementation with the stiffness estimator described in §7.4.3. The scenario tested in each comparison is a vehicle tracking lane change maneuvers on surfaces varying from asphalt to snow. As stated in §7.4.2 and §7.4.3 the primary constraints of concern are that the vehicle stays on the road; however future work could incorporate more complicated obstacle avoidance constraints. The metrics used to evaluate the controllers are cost and score, and are computed as follows:

$$\text{Cost} = \sum_k J(z_k, \tilde{u}_k), \tag{7.44}$$
$$\text{Score} = \sum_k (\max(y_k - y_{\max}, 0) + \max(y_{\min} - y_k, 0))\tau_{\text{plan}}. \tag{7.45}$$

7.5.1 Stiffness Estimator

This section evaluates the MPC formulation §7.4.2 with the stiffness estimator §7.3.1. The maneuver is a sequence of nine single lane-change maneuvers similar to the standardized IS0 3888-1 [ISO02] lane-change maneuver, with the middle three on snow and the rest on dry asphalt. To investigate the learning behavior of the controller, the surface change occurs during a straight portion, where the stiffness is unobservable. The reference velocity is fixed to 17 m/s. The reference

is generated with Bezier polynomials and the position, heading, longitudinal velocity, and yaw rate are given to the controllers to track. The lateral constraints enforced are that the vehicle is not allowed to leave the road boundaries. The simulation model uses the Pacejka tire model described in §7.2. The Pacejka parameters for each road surface are randomly perturbed at each controller timestep, with samples drawn from a uniform distribution up to $\pm$ 5% for asphalt and 10% for snow. The following 5 NMPC controllers are compared:

1. STOCHASTIC: proposed Chance-constrained NMPC with online adaptation to stiffness-estimation results.
2. ADAPTIVE: nominal NMPC with online adaptation to the mean cornering stiffness.
3. SNOW: nominal NMPC with cornering stiffness fixed to snow parameter values.
4. ASPHALT: nominal NMPC with cornering stiffness fixed to dry asphalt parameter values.
5. ORACLE: NMPC with true mean, nonlinear tire-force model.

The ORACLE controller is included to provide a lower bound on cost and constraint violations for the simulations. Its performance cannot be achieved in practice because it is given the exact tire force curve used by the simulation model; in reality there will be model mismatch due to inaccuracies in both the tire force and single-track vehicle models.

All controllers perform 1 SQP iteration per time step [GZQ+20b] and the nominal NMPC controllers 2-5 do not have stochastic constraints. For the stability constraints in (7.38d), the ASPHALT and SNOW controllers assume road friction values of $\mu = 1.0$ and 0.35, respectively. Since the ORACLE utilizes a nonlinear tire model, the stability constraints (7.38d) are not enforced. The least-squares cost (7.32) prioritizes the lateral position and wheel speed inputs. A timestep of $\tau_{\text{plan}} = 0.05$ with a prediction horizon of 2 s is used. The stiffness estimator is run at 100 Hz. The constraint satisfaction probability for the STOCHASTIC controller is set to 95%, i.e., $\Delta = 0.05$. The results of 200 trials are shown in Table 7.1. In most trials, the ASPHALT controller destabilizes the vehicle and the trials were terminated early; the reported cost and score is summed up to the point of termination.

Figure 7.6 (a) shows the trajectories, and Figure 7.7 shows the stability constraints in (7.38d) for an example trial. The ASPHALT controller is unable to safely navigate the maneuvers on snow; whereas the SNOW controller behaves conservatively on asphalt. The STOCHASTIC and ADAPTIVE controllers overshoot the first maneuver on snow, but are able to match the performance of the SNOW and ORACLE controllers once they have learned about the surface change. The average cost for the STOCHASTIC controller is 1% less than the ADAPTIVE controller, 86% less than the SNOW controller, and only 29% more than the ORACLE. The STOCHASTIC controller does not

Table 7.1: Metrics of 200 random trials of single lane change maneuvers on Dry Asphalt/Snow at 17 m/s with linear tire force controllers

NMPC Controller	Cost (7.44)		Score (7.45)	
	mean	max	mean	max
STOCHASTIC	0.339	0.448	0	0
ADAPTIVE	0.342	0.480	1.4e-4	0.013
SNOW	2.463	2.500	0	0
ASPHALT	136.0	562.8	3.254	20.70
ORACLE	0.263	0.268	0	0

Table 7.2: Metrics of 200 random trials of single lane change maneuvers on Dry Asphalt/Snow at 19 m/s with linear tire force controllers

NMPC Controller	Cost (7.44)		Score (7.45)	
	mean	max	mean	max
STOCHASTIC	1.193	1.733	1.2e-3	0.037
ADAPTIVE	1.814	3.881	0.021	0.086
SNOW	3.329	3.442	0.034	0.102
ASPHALT	219.7	834.8	5.92	16.15
ORACLE	0.710	0.725	0	0

violate the lateral constraints in this example, and performs better on the score metric than the ADAPTIVE controller. The average cost for the STOCHASTIC controller is within 30% of the cost for the ORACLE.

The second case study uses the same setup, except the speed is increased to 19 m/s. The results of 100 trials are shown in Table 7.2. Compared to the previous case study, all of the controllers have an increased cost and, aside from the ORACLE, some constraint violations. The SNOW controller frequently violates the lateral constraints due to the fact that it is using a linear tire model with fixed stiffness parameters and the tires saturate at the faster velocity. The ADAPTIVE controller violates the lateral constraints frequently during the first snow maneuver, since it does not take uncertainty in the stiffness estimate into account while it is learning the surface change. The average score for the STOCHASTIC controller is 94% less than the ADAPTIVE controller and 96% less than the SNOW controller. The average cost for the STOCHASTIC controller is 34% less than the ADAPTIVE controller, 64% less than the SNOW controller, but now 68% more than the ORACLE. The maximum cost for the ADAPTIVE controller also increases significantly, relative to the STOCHASTIC. The average cost for the STOCHASTIC controller is now within 67% from the cost for the ORACLE.

Overall, the results show that the STOCHASTIC controller is able to closely match the performance of the ORACLE controller once it has learned about the road surface. Incorporating stochas-

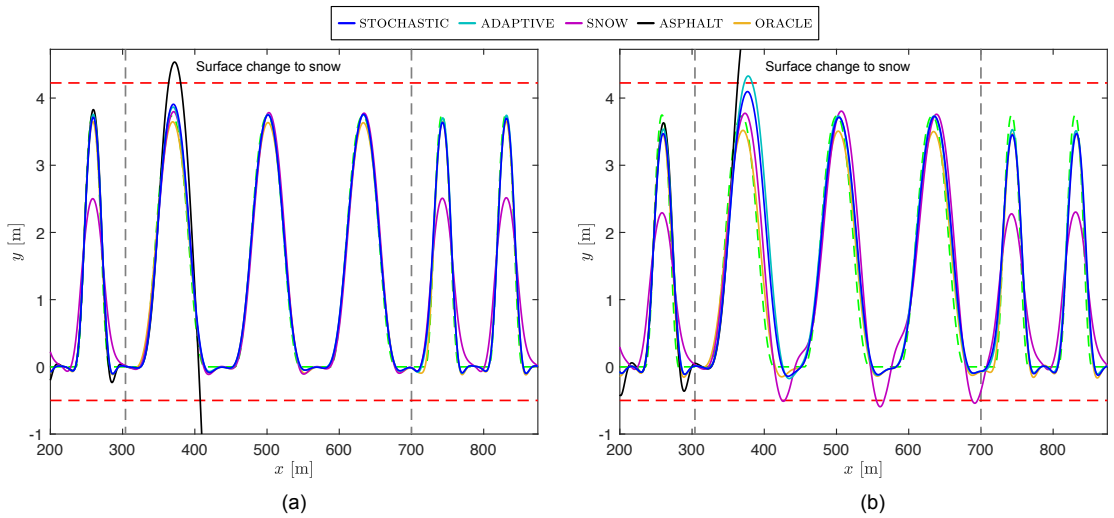


Figure 7.6: Position trajectories for a sample trial at 17 m/s (subfigure (a)) and 19 m/s (subfigure (b)) where the middle 3 maneuvers are on snow and the others on dry asphalt. Red and green dashed lines are the constraints and reference. The gray dashed lines indicate the surface changes. The STOCHASTIC controller is able to satisfy the lateral constraints and closely match the performance of the ORACLE controller after it learns about the surface change. The snow controller violates the constraints at 19 m/s due to tire saturation, but the adaptive controllers are able to compensate.

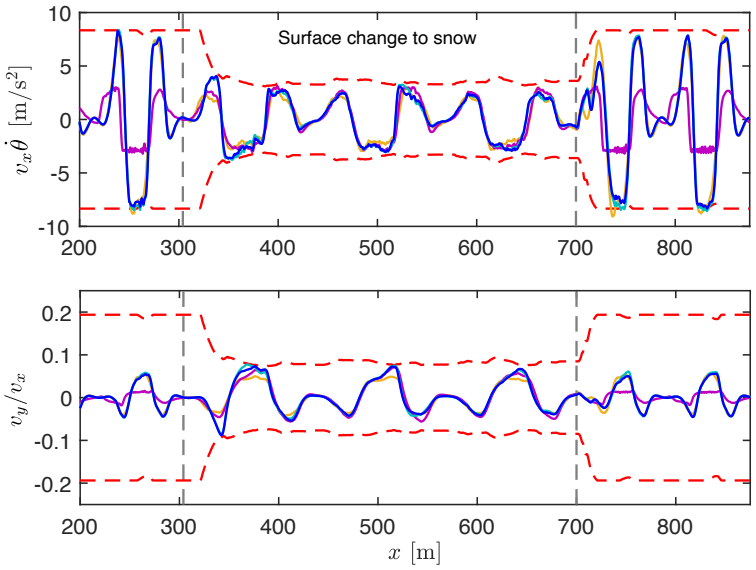


Figure 7.7: Stability constraints for the sample trial in Figure 7.6 (a), where the middle portion is on snow. Red dashed lines are the constraint boundaries, where the road friction is calculated with (7.39) using the estimator output from the STOCHASTIC controller. The constraints tighten during the snow portion. Coloring for the controllers is the same as in Figure 7.6. The ASPHALT controller destabilizes the vehicle and is omitted for clarity.

tic constraints improves safety and robustness during the learning portion and when tire saturation occurs. The constraint violations and cost for the STOCHASTIC controller are incurred mainly during the first snow maneuver, as the surface change occurs during a straight portion where the cornering stiffness is not observable.

7.5.2 Nonlinear

This section evaluates the MPC formulation §7.4.3 with the estimator from §7.3.2. This section considers a sequence of nine double lane-change maneuvers similar to the standardized IS0 3888-2 [ISO02] double lane-change maneuver, with the middle three on snow and the rest on dry asphalt. The maneuver is significantly more aggressive than the maneuver in §7.5.1, since the longitudinal length is similar, but the lateral distance for each lane change is a double lane change as oppose to a single one. To investigate the learning behavior of the controller, the surface change occurs during a straight portion, where the friction curve is unobservable. The reference is generated with Bezier polynomials and the position, heading, longitudinal velocity, and yaw rate are given to the controllers to track. For simplicity, the longitudinal velocity reference is constant, $v_{x,\mathrm{ref}} = 16$ m/s. The tire-friction estimator uses $n = 200$ particles and $m = 10$ basis functions, exploiting antisymmetry. The initial estimates and the different tuning parameters in the estimator are fairly generic, and the same as in [Ber20].

The following controllers are evaluated:

1. STOCHASTIC: proposed Chance-constrained NMPC method with online adaptation to the tire force estimator
2. ADAPTIVE: nominal NMPC with online adaptation to the mean function from thetire force estimator
3. SNOW: nominal NMPC that uses fixed snow Pacejka constants
4. ASPHALT: nominal NMPC that uses fixed asphalt Pacejka constants
5. ORACLE: nominal NMPC with the true, mean Pacejka constants

As in §7.5.1, the ORACLE controller is included to provide a lower bound on cost and constraint violations for the simulations. Its performance cannot be achieved in practice because it is given the exact tire force curve used by the simulation model; in reality, there will be model mismatch due to inaccuracies in both the tire force and single-track vehicle models. All of the controllers perform 1 SQP iteration per planning instance. The estimators are executed at 100 Hz and the different MPCs at 20 Hz, with a prediction horizon of 2 s. The constraint satisfaction probability

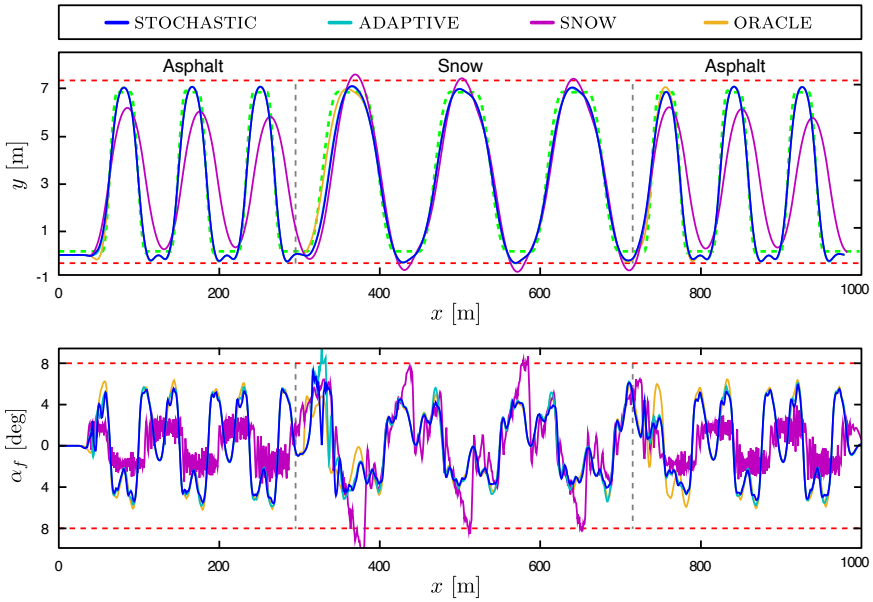


Figure 7.8: Resulting path for the various approaches during the lane-change maneuvers. Red and green dashed lines are the constraints and reference, respectively. The solid lines indicate the trajectory for each controller. The snow controller is unable to satisfy the lateral constraints due to mismatch between the MPC and simulation models.

for STOCHASTIC is set to 95 %, i.e., $\Delta = 0.05$. The least-squares cost (7.32) prioritizes the lateral position. In all of the simulations for the considered vehicle control maneuver, ASPHALT diverged after the transition to snow; hence, it is omitted from the results section.

Fig. 7.8 shows a comparison of the lateral tracking performance between different controllers. The results demonstrate that the reference trajectory is tracked well by the STOCHASTIC, ADAPTIVE and ORACLE controllers, and that there are some transients from the adaptive ones initially due to the convergence time of the tire-friction estimator. The SNOW controller performs conservatively in the asphalt sections, and also has difficulty tracking the reference in the snow portion; due to the aggressive nature of the maneuver and inaccuracies between the MPC and simulation model. By investigating the path in Figure 7.8 alone (top plot), there does not seem to be a clear benefit of adding uncertainty prediction to the MPC formulation. However, when also investigating the front-wheel slip angle for the same comparison (bottom plot), the constraint violations for the ADAPTIVE MPC are clearly visible during the initial transition to snow. In the plots one can also see that on the transition to asphalt from snow, the adaptive controllers are initially conservative, but match the performance of the oracle once they learn about the surface change. The tracking performance, while interesting, is only one component of the cost function, and a more objective measure is to analyze how the cost (7.44) and score (7.45) compare for different noise realizations. To this end, Table 7.3 shows the results for 200 Monte-Carlo runs. The mean cost over the runs is

Table 7.3: Metrics of 200 random trials of double lane change maneuvers on Dry Asphalt/Snow at 16 m/s with nonlinear tire force controllers

NMPC Controller	Cost (7.44)		Score (7.45)	
	mean	max	mean	max
STOCHASTIC	1.727	1.876	5.0e-3	0.022
ADAPTIVE	1.693	10.589	0.028	5.389
SNOW	14.906	17.475	0.838	2.357
ORACLE	1.091	1.115	0	0

similar between STOCHASTIC and ADAPTIVE, but the worst-case cost and both score metrics are larger when not accounting for the uncertainty.

7.5.3 Discussion

The results in §7.3.1 and §7.3.2 show the benefits of incorporating adaptivity and the uncertainty associated with the vehicle model in the trajectory optimization formulation. From these results it appears that using the nonlinear estimator and tire model will allow the vehicle to perform more aggressive maneuvers, however I have shown that the linear estimator and tire model is able to compensate for tire saturation in moderately aggressive maneuvers. But it is important to note that, when operating at low slip angles, the linear tire force controller will be overly optimistic about the available tire force at higher slip angles, which can lead to the vehicle planning aggressive actions at future timesteps in the planning horizon. Even if the vehicle is able to stabilize itself after making such errors, this can be problematic if constraint violations occur. A benefit of the linear estimator compared to the GP-based nonlinear estimator, is that the linear estimator is learning a parameter that parameterizes the entire tire force curve; which can improve convergence rates. Although the nonlinear estimator leverages the anti-symmetric nature of tire models, when transitioning from lower friction surfaces to higher friction surfaces, the controller behaves conservatively as there is large uncertainty in what the tire forces at higher slip angles are.

For both estimators and controllers, improving the design of the reference trajectory to encourage persistent excitation, or modulating the forgetting factor in the estimator after a surface change is detected, could greatly improve their performance. Additionally, if extrinsic sensors are available, incorporating an algorithm that forecasts changes in road surface and incorporates them into the MPC program could improve performance. Such a forecast could also be used to modulate the forgetting factor in each estimator to speed up convergence during surface changes; however the tradeoff between this and increasing the noise of the estimates must be managed. In the case of the adaptive constraints for the linear controller, the approximation of the road friction in the stability constraints can also be improved by using a more sophisticated estimation algorithm in

combination with extrinsic sensing. Finally, developing a planning framework that is able to leverage information from the tire force estimators to adjust the reference trajectory would offer great benefits in an autonomous application.

CHAPTER 8

Conclusion

This dissertation has made contributions to enable safe receding-horizon trajectory planning for mobile robots. Recall that there were three major challenges this dissertation aimed to address. First collision checks are typically performed at discrete time steps, which introduces an unideal tradeoff between runtime and safety; since a more accurate (frequent) collision check takes longer to execute. Second, using simplified dynamic models of the robot allow for faster optimization, however there can be a nontrivial gap between the model used and the actual system. Finally, there is unavoidable uncertainty in the motion of other agents or robots. This chapter provides a summary of the contributions then discusses future research directions.

8.1 Summary of Contributions

This section reviews the contributions of this dissertation. In Chapter 3, I developed a method using Sums-of-Squares (SOS) programming for representing the forward reachable set of a dynamic model tracking parameterized trajectories as a polynomial level set. Importantly the SOS program allows one to capture the reachable set over time intervals, addressing the first challenge. The program also allows for the model to contain an affinely-appearing disturbance term; which addresses the second challenge robustly. Chapter 4 describes Reachability-based Trajectory Design (RTD) the method for using the reachable sets for online trajectory optimization. I presented an obstacle discretization method that represents static and dynamic obstacles as sets of discrete points in a provably safe manner; hence the online trajectory planner is a polynomial optimization program. The discretization method can be applied to arbitrary polygon predictions, so any method of generating conservative predictions of this form can be incorporated to address the third challenge robustly. Chapter 5 provides demonstrations and comparison results for RTD against the state-of-the-art trajectory planners.

Chapters 4 and 5 operate under the assumption that the third challenge is addressed robustly by providing the planner with conservative predictions of obstacles' and their motion. In many

applications it is useful to consider obstacle predictions that are represented by probability functions. To address this, Chapter 6 presents the Chance-constrained Parallel Bernstein Algorithm (CCPBA), a branch-and-bound algorithm for solving chance-constrained polynomial optimization programs. CCPBA allows one to address the second challenge robustly and the third challenge probabilistically by solving a chance-constrained version of the RTD trajectory optimization program where the probability of collision is limited. Chapter 7 provides contributions to address the second challenge probabilistically. Again, the ego robot plans with a dynamic model with an affinely-appearing disturbance term; however instead of robustly accounting for all possible disturbances, the value of the disturbance is estimated online as a probability function, and incorporated into a chance-constrained Model Predictive Control program. I presented an application of an autonomous vehicle driving on varying road surfaces, where the tire force is estimated using both linear and nonlinear representations.

8.2 Future Work

There are several possible areas for future work to build on and improve the contributions in this dissertations. I now discuss these gaps and potential ways forward.

Sums-of-Squares Programming The Sums-of-Squares program presented for reachable set computation in Chapter 3 is memory intensive, hence the dimension of the dynamic model and trajectory parameter space has been limited to less than 6 in the presented applications, and the polynomials have been limited to a degree of 10 or less. The limited dimension for the dynamic model introduces conservatism since higher-dimensional affects are treated as uncertainty. The limited dimension for the planning space reduces the richness of possible actions for the robot. Additionally, in the receding horizon implementation, the ability to add parameters for the currently applied trajectory parameters would eliminate the need for bounding error associated with predicting the future robot state in Assumption 11. I presented a system decomposition method to alleviate this somewhat in § 3.4, however this seemingly allows us to add only one or two more dimensions. To increase the size of the SOS programs I suggest investigating the use of conic solvers to solve the reachability programs [PY19], that offer memory savings compared the to the semidefinite programming implementation used in this dissertation.

Tracking Error Computation The functions used to bound the gap between the planning model and robot are currently computed offline via sampling. Methods using reachability analysis and SOS programming should also be developed to compute these functions. Tangentially one may consider optimizing the design of the lower level controller in conjunction with the reachable set,

using methods similar to [MVTT14, MT17].

Dimension of CCPBA One disadvantage of CCPBA is that the decision variable space and uncertainty vector space are limited, and the number of patches and associated computations increases exponentially with dimension. To this end, future research on efficient splitting strategies to evaluate the chance constraints, possibly drawing on theory from algorithms for adaptive, numerical integration [DR07] should be conducted.

Time Discretization in CCPBA The presented implementation of CCPBA performs a collision check at discrete timepoints and uses Boole's inequality to account for interdependence between timepoints. This is unfortunate since there is no formal guarantee of collision avoidance between timesteps, but it is known that the action of Boole's inequality is overly conservative and becomes more conservative as one increases the discretization fineness. Determining a method to account for collision checking over continuous time intervals for chance-constrained programming should be studied. One could adapt an approach similar to [JHW21], but use the Bernstein method for chance constraint evaluation as oppose to Cantelli's inequality.

Bernstein Polynomials for other Risk Measures Bernstein polynomials may also prove useful in the evaluation of different types of risk metrics. Recent methods in risk bounded planning have explored the conditional value-at-risk (CVaR) metric [HY20, CBS+21], other have proposed similar metrics for constraint violation [NPDC+21]. These differ from the chance-constrained program in that they seek to limit the severity of constraint violation or a high-cost, but rare event occurring. Direct evaluation of these constraints usually involves the expectation of the maximum of a random variable and 0. Bernstein polynomials proved useful for bounding the expectation of an indicator function (6.3); so extending them to evaluate the expectations with these types of maximums is a possibility.

Forecasting of Uncertainty for Adaptive NMPC The adaptive, chance-constrained approach in Chapter 7 is reactive in a sense that it assumes the current estimate of uncertainty is true for the time horizon. Estimating a property like the road friction can be done with extrinsic sensors [KET17], and a forecast of the uncertainty could be incorporated into an NMPC program. [CWHL21] incorporates a given friction estimate into a deterministic NMPC program. However research can be done on how to generate such an estimate in a way that can be incorporated into a chance-constrained program.

APPENDIX A

Reachable Set Computation

A.1 Alternative Method for Incorporating the Footprint

Program (D_T) as written in §3.3 incorporates the footprint of the robot into the reachability computation by augmenting the size position states in the initial condition set Z_0. This is problematic since in (D_T) and (D) the dynamics (3.13) get applied to every point in Z_0; hence they neglect the fact that the robot is a rigid body.

This section presents an alternate formulations and programs to compute $\mathcal{X}_{\text{TFRS}}$ and $\mathcal{X}_{\text{FRS}}$ that preserve the rigidity of the robot's body. First assume that the footprint is given by the collection of circles, and perform a collision check against a point obstacle, whose state is $x \in X \subset \mathbb{R}^2$ as in Assumption 78.

Example 85. *The footprint of a sedan can be approximated as 3 circles spaced across the longitudinal axis as shown in Figure A.1. The functions defining $\mathcal{A}_{\text{col}}$ are then*

$$\mathcal{A}_{\text{col}} = \left\{ (z, x) \in Z \times X \;\middle|\; \bigvee_{i \in \{1,2,3\}} h_{\text{col}}^{(i)}(z, x) \geq 0 \right\}, \tag{A.1}$$

$$h_{\text{col}}^{(i)}(x, z) = r^2 - (x_1 - z_1 - l_i \cos z_3)^2 + (x_2 - z_2 - l_i \sin z_3)^2 \tag{A.2}$$

where l_i gives the center of each circle along the longitudinal axis of the vehicle and r is the radius of each circle and $[z_1, z_2, z_3]^\top$ is xy position and heading of the ego robot.

Next define the FRS at a point in time, considering the robot footprint.

$$\begin{aligned} \mathcal{X}_{\text{TFRS}} = \Big\{ & (t, x, k) \in \ [0, T] \times X \times K \mid \exists\ z_0 \in Z_0, \text{ and } d \in L_d \\ & \text{s.t. } z = \tilde{z}(t),\ (z, x) \in \mathcal{A}_{\text{col}}, \\ & \text{where } \dot{\tilde{z}}(\tau) = f(\tau, \tilde{z}(\tau), k) + g(\tau, \tilde{z}(\tau), k) \circ d(\tau) \\ & \text{a.e. } \tau \in [0, T] \text{ and } \tilde{z}(0) = z_0 \Big\}. \end{aligned} \tag{A.3}$$

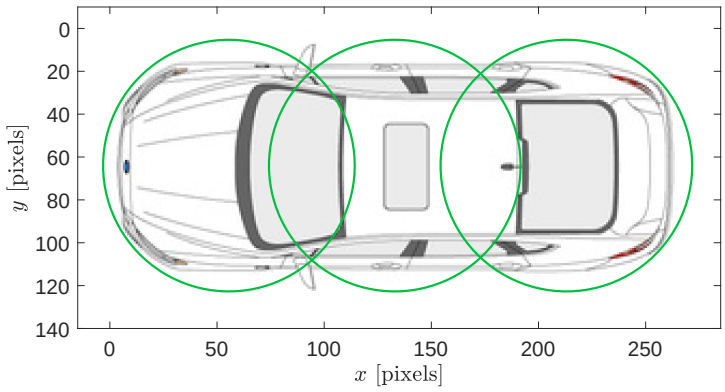


Figure A.1: Footprint of a passenger sedan modeled by 3 circles spaced along the longitudinal axis.

Then a function $v : [0,T] \times X \times K \to \mathbb{R}$, whose 0 super-level set contains the footprint can be found by solving the following program

$$
\begin{aligned}
\inf_{v,\tilde{v},w,q} \quad & \int_{[0,T]\times X\times K} w(t,x,k)\, d\lambda_{[0,T]\times X\times K} && (D_{FT}) \\
\text{s.t.} \quad & \mathcal{L}_f \tilde{v}(t,z,k) + q(t,z,k) \leq 0, && (D_{FT}1) \\
& \mathcal{L}_g \tilde{v}(t,z,k) + q(t,z,k) \geq 0, && (D_{FT}2) \\
& -\mathcal{L}_g \tilde{v}(t,z,k) + q(t,z,k) \geq 0, && (D_{FT}3) \\
& q(t,z,k) \geq 0, && (D_{FT}4) \\
& -\tilde{v}(0,z,k) \geq 0, && (D_{FT}5) \\
& v(t,x,k) + \tilde{v}(t,z,k) \geq 0, && (D_{FT}6) \\
& \text{on } \{(t,z,x,k) \in [0,T] \times Z \times X \times K \mid h^{(i)}_{\text{col}}(z,x) \geq 0\}, \\
& w(t,x,k) \geq 0, && (D_{FT}7) \\
& w(t,x,k) - v(t,x,k) - 1 \geq 0, && (D_{FT}8)
\end{aligned}
$$

where $(D_{FT}6)$ consists of $n_{h\text{col}}$ constraints each enforced on the 0 superlevel sets of $h^{(i)}_{\text{col}}$. Constraints $(D_{FT}1), (D_{FT}2), (D_{FT}3), (D_{FT}4)$ apply for all $(t,z,k) \in [0,T] \times Z \times K$. Constraint $(D_{FT}5)$ applies for all $(z,k) \in Z_0 \times K$. Constraints $(D_{FT}7)$ and $(D_{FT}8)$ are enforced on $[0,T] \times X \times K$. The infimum is taken over $(v,w,q) \in C([0,T] \times X \times K) \times C^1([0,T] \times Z \times K) \times C([0,T \times X \times K) \times C([0,T] \times Z \times K)$. The following lemma relates the solutions of (D_{FT}) and the "footprint" FRS at a point in time.

Lemma 86. *Let* $(v, \tilde{v}, w, q)$ *be a feasible solution to* (D_{FT}) *then*

$$(t, x, k) \in \mathcal{X}_{\text{TFRS}} \implies v(t, x, k) \geq 0 \tag{A.4}$$

Proof. Lemma 15 ensures that $\tilde{v}$ is negative along trajectories produced by the trajectory tracking model. Constraint $(D_{FT}6)$ provides that v is positive when $\tilde{v}$ is negative and $(z, x) \in \mathcal{A}_{\text{col}}$. □

A similar Lemma as 16 can also be proven to provide convergence of w to an indicator function on $\mathcal{X}_{\text{TFRS}}$. I also provide an alternate definition and program to compute the FRS over the time horizon

$$\begin{aligned}\mathcal{X}_{\text{FRS}} = \Big\{ & (x, k) \in X \times K \mid \exists\, z_0 \in Z_0,\ t \in [0, T] \text{ and } d \in L_d \\ & \text{s.t. } z = \tilde{z}(t),\ (z, x) \in \mathcal{A}_{\text{col}}, \\ & \text{where } \dot{\tilde{z}}(\tau) = f(\tau, \tilde{z}(\tau), k) + g(\tau, \tilde{z}(\tau), k) \circ d(\tau) \\ & \text{a.e. } \tau \in [0, T] \text{ and } \tilde{z}(0) = z_0 \Big\},\end{aligned} \tag{A.5}$$

and a program to compute a function $w : X \times K \to \mathbb{R}$, whose 1 super-level set contains $\mathcal{X}_{\text{FRS}}$:

$$\begin{aligned}
\inf_{v, \tilde{v}, w, q} \quad & \int_{X \times K} w(x, k)\, d\lambda_{X \times K} && (D_F) \\
\text{s.t.} \quad & \mathcal{L}_f \tilde{v}(t, z, k) + q(t, z, k) \leq 0, && (D_F1) \\
& \mathcal{L}_g \tilde{v}(t, z, k) + q(t, z, k) \geq 0, && (D_F2) \\
& -\mathcal{L}_g \tilde{v}(t, z, k) + q(t, z, k) \geq 0, && (D_F3) \\
& q(t, z, k) \geq 0, && (D_F4) \\
& -\tilde{v}(0, z, k) \geq 0, && (D_F5) \\
& v(t, x, k) + \tilde{v}(t, z, k) \geq 0, && (D_F6) \\
& \text{on } \{(t, x, z, k) \in [0, T] \times X \times Z \times K \mid h^{(i)}_{\text{col}}(x, z) \geq 0\}, && \\
& w(x, k) \geq 0, && (D_F7) \\
& w(x, k) - v(t, x, k) - 1 \geq 0, && (D_F8)
\end{aligned}$$

where (D_F6) consists of $n_{h\text{col}}$ constraints each enforced on the 0 superlevel set of $h^{(i)}_{\text{col}}$. Constraints $(D_F1), (D_F2), (D_F3), (D_F4)$ apply for all $(t, z, k) \in [0, T] \times Z \times K$. Constraint (D_F5) applies for all $(z, k) \in Z_0 \times K$. Constraint (D_F7) is enforced on $X \times K$. The infimum is taken over $(v, w, q) \in C([0, T] \times X \times K) \times C^1([0, T] \times Z \times K) \times C(X \times K) \times C([0, T] \times Z \times K)$.

Lemma 87. *Let* $(v, \tilde{v}, w, q)$ *be a feasible solution to* (D_F) *then* $(x, k) \in \mathcal{X}_{\text{FRS}} \implies w(x, k) \geq 1$.

Proof. Lemma 15 ensures that $\tilde{v}$ is negative along trajectories produced by the trajectory tracking model. Constraint $(D_F 6)$ provides that v is positive when $\tilde{v}$ is negative and $(z, z) \in \mathcal{A}_{\text{col}}$. Constraint $(D_F 8)$ provides that w is greater than 1 when v is positive at any point in time $t \in [0, T]$. $\square$

A similar Lemma as 16 can also be proven to provide convergence of w to an indicator function on $\mathcal{X}_{\text{FRS}}$. Similar to how I treated the time intervals in §3.5 one may also solve 2 separate programs by solving (D_{FT}) first to obtain $\tilde{v}$, then formulating a smaller program with constraints $(D_{FT}6) - (D_{FT}8)$. All of the programs presented in this section can also be implemented via Sums-of-Squares (SOS) programming as in §3.3.2 provided the Assumption 78 is satisfied.

A.2 FRS Program for Bounded State Tracking Error

If instead of searching for a tracking error bound on the dynamics, one is constructed for the states, as in Remark 10, an alternate version of (D) can be solved to compute the FRS. First restate the remark, then give the FRS program.

Remark 10. *Consider a trajectory-producing model described by open-loop dynamics, for example as in Examples 4 and 5. One can modify* (3.12) *Assumption 7 to be*

$$\max_{z_{\text{hi}} \in Z_{\text{hi},0}} |z_{\text{hi},i}(t; z_{\text{hi},0}, k) - z(t; z, k)| \leq G_i(\tau, k)$$

Such a tracking error bound is less conservative, as it takes into account the integrated tracking error. For example, second order controllers may produce oscillations around a setpoint, the error bounds produced by (3.12) *will lead to conservative FRSs in this case. To incorporate such a bound into the reachable set computation, one can simply set* $g(\cdot) = \frac{d}{dt} G(t, k)$ *or solve* (A.6).

The FRS program can be given by

$$\begin{aligned}
\inf_{v,w} \quad & \int_{X \times K} w(x, k)\, d\lambda_{X \times K} && \text{(A.6)} \\
\text{s.t.} \quad & \mathcal{L}_f v(t, z, k) \leq 0, && (A.6.1) \\
& -v(0, z, k) \geq 0, && (A.6.2) \\
& w(x, k) + v(t, z, k) - 1 \geq 0, && (A.6.3) \\
& \text{on } \{(t, x, z, k) \in [0, T] \times X \times Z \times K \mid h_G(t, x, z, k) \geq 0\}, \\
& w(x, k) \geq 0, && (A.6.4)
\end{aligned}$$

where

$$h_G(t,x,z,k) = (x - z + G(t,k))(x + G(t,k) - z). \tag{A.7}$$

Constraint (A.6.1) applies for all $(t,z,k) \in [0,T] \times Z \times K$. Constraint (A.6.2) applies for all $(z,k) \in Z_0 \times K$. Constraint (A.6.4) applies for all $(x,k) \in \mathcal{X} \times K$. The infimum is taken over $(v,w) \in C^1([0,T] \times Z \times K) \times C(X \times K)$.

Lemma 88. *Let (v,w) be a feasible solution to* (A.6)*. Let $z_{\text{hi}}(t; z_{\text{hi},0}, k)$ denote the solution to the high-fidelity* (3.4) *at time t beginning from $z_{\text{hi},0} \in \{z_{\text{hi}} \in Z_{\text{hi}} \mid \text{proj}_X(z_{\text{hi}}) \in X_0\}$ under control input u_k. For every $t \in [0,T]$, $k \in K$, and $z_{\text{hi},0} \in \{z_{\text{hi}} \in Z_{\text{hi}} \mid \text{proj}_X(z_{\text{hi}}) \in X_0\}$, $w(\text{proj}_X(z_{\text{hi}}), k) \geq 1$.*

Proof. $v(t,z,k) \leq 0$ for any z produced by the trajectory producing model (3.2) via Lemma 15. Constraint (A.6.3) and (A.7) provide that if $v(t,z,k)$ is negative and z_{hi} and z satisfy (3.16) then $w(\text{proj}_X(z_{\text{hi}}), k) \geq 1$. □

A.3 Program for Collision Functions

This section outlines a program to compute h_{col} in Assumption 78. Note that it can also be appended to the reachability program in §A.1. Let $z \in Z$ described the state of the ego robot and $\zeta \in Z_{\text{obs}}$ describe the state of the obstacle. Let $x \in X$ described 2D position coordinates. Let the ego vehicle be described by semialgebraic sets

$$\mathcal{A}_{\text{ego}} = \{(x,z) \in X \times Z \mid h^{(i)}_{\text{ego}} \geq 0, \text{ for any } i \in \{1,\ldots,n_{\text{ego}}\}\}, \tag{A.8}$$

where it is understood that if a particular $h^{(j)}_{\text{ego}}$ is a vector of polynomials, then all $h^{(j)}_{\text{ego}}(x,z) \geq 0$ for the index j to count as non-negative. Let the obstacle be described by semialgebraic sets

$$\mathcal{A}_{\text{obs}} = \{(x,\zeta) \in X \times Z_{\text{obs}} \mid h^{(i)}_{\text{obs}} \geq 0, \text{ for any } i \in \{1,\ldots,n_{\text{obs}}\}\}. \tag{A.9}$$

For example consider the intersection of the ego vehicle described in Example 85 and a rectangular obstacle aligned with the road frame as in §6.3.3. h_{ego} will be defined as in (A.2), h_{obs} can be defined as

Example 89. *A rectangle of width W and length L can be defined with*

$$h_{obs}(x,\zeta)^{(1)} = \begin{bmatrix} (x_1 - \zeta_1 + L/2)(\zeta_1 + L/2 - x_1) \\ (x_2 - \zeta_2 + W/2)(\zeta_2 + W/2 - x_2) \end{bmatrix} \tag{A.10}$$

Then a function $h_{\text{col}} : Z \times Z_{\text{obs}} \to \mathbb{R}$ with the property

$$\{\exists x \in X \mid (x, z) \in \mathcal{A}_{\text{ego}},\ (x, \zeta) \in \mathcal{A}_{\text{obs}}\} \implies h_{\text{col}}(z, \zeta) \geq 0 \tag{A.11}$$

can be found by the following program

$$\begin{aligned}
\inf_{v_{\text{ego}}, v_{\text{obs}}, h_{\text{col}}, w} \quad & \int_{Z \times Z_{\text{obs}}} w(z, \zeta)\, d\lambda_{Z \times Z_{\text{obs}}} && (A.12)\\
\text{s.t.} \quad & v_{\text{ego}}(x, z) \geq 0, \text{ on } \mathcal{A}_{\text{ego}} && (A.12.1)\\
& v_{\text{obs}}(x, \zeta) \geq 0, \text{ on } \mathcal{A}_{\text{obs}} && (A.12.2)\\
& h_{\text{col}}(z, \zeta) - v_{\text{ego}}(x, z) - v_{\text{obs}}(x, \zeta) \geq 0, && (A.12.3)\\
& w(z, k\zeta) - h_{\text{col}}(z, \zeta) - 1 \geq 0, && (A.12.4)\\
& w(z, \zeta) \geq 0, && (A.12.5)
\end{aligned}$$

where constraint (A.12.3) applies for all $(x, z, \zeta) \in X \times Z \times Z_{\text{obs}}$ and constraints (A.12.4, A.12.5) apply for all $(z, \zeta) \in Z \times Z_{\text{obs}}$. The infimum is taken over $C(X \times Z) \times C(X \times Z_{\text{obs}}) \times C(Z \times Z_{\text{obs}}) \times C(Z \times Z_{\text{obs}})$. This program can be implemented with Sums-of-Squares polynomials if the sets defining the footprints are semilagebriac. Note that in cases such as Example 89, where the set is analytically defined you could forgo the inclusion of v_{obs} and constraint (A.12.2), and enforce (A.12.3) on $\{(x, \zeta) \mid h^{(1)}_{\text{obs}}(x, \zeta) \geq 0\}$ additionally. However in cases with more than 1 set defining the footprint, it is advantages to include separate functions and constraints, since the constraint in (A.12.3) involve the introduction of s-functions from $C(X \times Z \times Z_{\text{obs}})$.

APPENDIX B

Discrete Obstacle Representation

This section provides a proof of Theorem 52, and begins by first introducing necessary definitions and Lemmas.

Definition 90. *Define a* transformation $R_t \in \mathsf{SE}(2)$. *Each R_t is given by a rotation angle $\theta_t \in [0, 2\pi)$ and a translation vector $s_t \in \mathbb{R}^2$, so R_t transforms a point $p \in \mathbb{R}^2$ as*

$$R_t p = \begin{bmatrix} \cos\theta_t & -\sin\theta_t \\ \sin\theta_t & \cos\theta_t \end{bmatrix} (p - c) + s_t + c, \tag{B.1}$$

where $c \in \mathbb{R}^2$ is the center of rotation (which is typically considered as the geometric center of X_0 when applying R_t to the robot). The subscript indicates that the transformation is indexed by time $t \in [0, T]$. Define a transformation family $\{R_t \mid t \in [0, T]\}$ *of planar translations and rotations that is continuous with respect to t.*

Next, define useful geometric objects.

Definition 91. *Let $I \subset \mathbb{R}^2$ be a* line segment*, also called an* interval *when it lies on either the x- or y-axis. Let $E_I = \{e_1, e_2\} \subset I$ denote the* endpoints of I*, such that I can be written as $I = \{e_1 + s \cdot (e_2 - e_1) \mid s \in [0, 1]\}$. The* length *of I is $\|e_1 - e_2\|_2$. Suppose I has a pair of distinct endpoints $\{e_1, e_2\}$, and create the set $\ell_I = \{e_1 + s \cdot (e_2 - e_1) \mid s \in \mathbb{R}\} \subset \mathbb{R}^2$, i.e. a line that passes through e_1 and e_2. ℓ_I is called* the line defined by I.

Definition 92. *Let $A \subset \mathbb{R}^2$ be a set with a boundary and $a_1, a_2 \in \partial A$. The line segment $\kappa = \{a_1 + s \cdot (a_2 - a_1) \mid s \in [0, 1]\}$ is a* chord *of A.*

Definition 93. *A* circle $C \subset \mathbb{R}^2$ *of radius $R \geq 0$ with center $p \in \mathbb{R}^2$ is given by the set $\{p' \in \mathbb{R}^2 \mid \|p' - p\|_2 = R\}$. An* arc $A \subset \mathbb{R}^2$ *is any connected, closed, strict subset of a circle; this means that any arc has two* endpoints $a, b \in \mathbb{R}^2$.

Definition 94. *Let $c \in X_0$ denote the center of mass of the robot's footprint at time 0. Let I be a line segment as in Definition 91 with two distinct endpoints $E_I = \{e_1, e_2\}$. Then $H_I \subset \mathbb{R}^2$ denotes the closed* half-plane defined by *I; this half-plane is determined by the line defined by I and by c as:*

$$H_I = \{p \in X \mid \operatorname{sign}(\delta_\pm(e_1, e_2, p)) = \operatorname{sign}(\delta_\pm(e_1, e_2, c))\}, \tag{B.2}$$

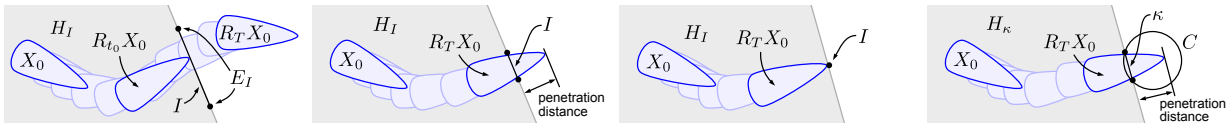
where $\operatorname{sign}(a) = 1$ for $a \geq 0$ and -1 otherwise. Now suppose that I is a line segment of length 0, i.e. $e_1 = e_2$, so one cannot directly define H_I as in (B.2)*. Suppose that $e_1 \neq c$. So, one can pick a point e' for which $(e' - e_1) \cdot (c - e_1) = 0$ where $\cdot$ denotes the standard inner product on $\mathbb{R}^2$, so the line segment from e_1 to c is perpendicular to the line segment from e_1 to e'. Then, H_I is given by* (B.2)*, but using e' in place of e_2.*

Next, define what it means for the robot's footprint to *pass through* and *penetrate* line segment, and provide Figure B.1 as an illustration.

Definition 95. *Let $I \subset X \setminus X_0$ be a line segment with endpoints E_I as in Definition 91, and H_I be the half-plane defined by I as in Definition 94. Suppose that the robot lies fully within H_I at time 0, i.e. $X_0 \subset \operatorname{int}(H_I)$. Let $\{R_t\}$ be a transformation family as in Definition 90. Let t_0, t_1 be indices in $(0, T]$ such that $R_t X_0$ intersects the "middle" of I, i.e. $R_t X_0 \cap (I \setminus E_I) \neq \emptyset$, for all $t \in [t_0, t_1]$. Furthermore, suppose that $R_t X_0 \subset H_I$ for all $t \in [0, t_0)$, and that no $R_t X_0$ can intersect the endpoints E_I (i.e. $R_t X_0 \cap E_I = \emptyset$) except at $t = T$. Such a transformation family attempts to* pass X_0 through I*. If X_0 is able to leave H_I while passing through I, i.e. $R_T X_0 \subset H_I^C$, then X_0 is said to* pass fully through I*.*

Definition 96. *Let $I \subset (X \setminus X_0)$ be a line segment as in Definition 91. Let H_I be the half-plane defined by I as in Definition 94, and suppose $X_0 \subset H_I$ strict. Let $\{R_t\}$ be a transformation family that attempts to pass X_0 through I by Definition 95. Suppose X_0 cannot pass fully through I, and that $R_T X_0 \cap H_I^C$ is nonempty, so there is some portion of X_0 that does pass through I. Consider all line segments perpendicular to I with one endpoint on I and the other at a point in $R_T X_0$ in H_I^C. The maximum length of any of these line segments is called the* penetration distance of X_0 through I*. The set $R_T X_0$* penetrates *I by this distance, as in Figure B.1b. If I is of length 0, then the penetration distance of X_0 through I is always 0, as in Figure B.1c.*

Definition 97. *Let $C \subset \mathbb{R}^2$ be a circle of radius R with center p as in Definition 93. Let X_0 be the robot's footprint at time 0 as in Definition 12. Let κ be a chord of C as in Definition 92. Then* passing X_0 into C through κ *is defined as passing X_0 through the chord κ as in Definition 95. If the length of κ is less than the width of X_0, then, by [Str82, Theorem 1], X_0 cannot pass fully through κ, but does penetrate the chord up to some distance as in Definition 96. Let H_κ be the*



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

Figure B.1: Passing through (as in Definition 95), penetrating (as in Definition 96), and penetrating into a circle (as in Definition 97). In each subfigure, a family $\{R_t\}_{t\in[0,T]}$ of continuous rotations and translations attempts to pass the convex, compact set X_0 through the line segment I with endpoints E_I. At $t = 0$, X_0 lies in the half-plane H_I, defined by I as in Definition 94. Each figure contains X_0 at its initial position R_0X_0 and final position R_TX_0 indicated by a dark outline. The lighter outlines between these positions show examples of X_0 being translated and rotated as $\{R_t\}$ is applied. In Figure B.1a, X_0 is able to pass fully through I; the index $t_0 \in [0, T]$ where X_0 first touches I is also shown with a dark outline. In Figure B.1b, X_0 is unable to pass fully through I, but penetrates through I by some distance into H_I^C. In Figure B.1c, the line segment I has length 0, so X_0 cannot pass through it, but instead stops as soon as it touches I, and achieves 0 penetration distance through I. Note that, in this case, H_I is defined by a line perpendicular to the line segment from I to the center of mass of X_0, as per Definition 94. In Figure B.1d, the circle C has a chord κ, and X_0 penetrates into C through κ by the penetration distance shown. The half-plane defined by κ is denoted H_κ.

closed half-plane defined by κ as in Definition 94. The penetration of X_0 into C through κ *is the maximum Euclidean distance from any point in $X_0 \cap C$ to a point in $X_0 \cap H_\kappa^C$.*

I next discuss how to find the point spacing, r, and arc point spacing, a.

Lemma 98. *Let $X_0 \subset \mathbb{R}^2$ be the robot's footprint at time 0, with width $\bar{r}$ (as in Definition 49). Let $\bar{b}$ be the maximum penetration depth corresponding to X_0 (as in Lemma 46). Pick $b \in (0, \bar{b})$. Then there exists $r \in (0, \bar{r}]$ such that, if I_r is a line segment of length r (as in Definition 91), and if $\{R_t\}$ is any transformation family that attempts to pass X_0 through I_r (as in Definition 95), then the penetration distance of X_0 through I_r (as in Definition 96) is less than or equal to b.*

Lemma 99. *Let X_0 be the robot's footprint at time 0 (as in Definition 12), with width $\bar{r}$ (as in Definition 49). Let $\bar{b}$ be the maximum penetration distance corresponding to X_0 (as in Lemma 46). Pick $b \in (0, \bar{b})$, and let $C \subset (X \setminus X_0)$ be a circle of radius b centered at a point $p \in X$ (as in Definition 93). Then there exists a number $a \in (0, \bar{r})$ such that, if κ_a is any chord of C of length a (as in Definition 92), then the penetration of X_0 into C through κ (as in Definition 97) is no larger than b.*

Theorem 52 is now proven.

Theorem 52. *Let X_0 be the robot's footprint at time 0 as in Definition 12, with width $\bar{r}$ as in Definition 49. Let $P \subset (X \setminus X_0)$ be a set of predictions as in Definition 31. Suppose that the*

maximum penetration depth $\bar{b}$ is found for X_0 as in Lemma 46. Pick $b \in (0, \bar{b})$, and find the point spacing r and the arc point spacing a. Construct the discretized obstacle X_p in Algorithm 2. Then, the set of all unsafe trajectory parameters corresponding to P is a subset of the trajectory parameters corresponding to X_p, i.e. $\pi_K(X_p) \supseteq \pi_K(P)$.

Proof. First show that any trajectory parameter outside of those corresponding to X_p cannot cause any point on the robot to enter the set P at any time $t \in [0, T]$. If no $q \in \pi_K(X_p)^C$ can cause a collision, then $\pi_K(X_p)^C \subseteq K_{\text{safe}}$, which implies that $\pi_K(X_p) \supseteq \pi_K(P)$. First, recall that the robot's high-fidelity model in (3.4) produces continuous trajectories (by Assumption 2) of the robot's footprint in $\mathbb{R}^2$, so the motion of the robot over the time horizon $[0, T]$ can be represented using a transformation family $\{R_t\}$ as in Definition 90.

Suppose $k \in \pi_K(X_p)^C$ is arbitrary, and the robot begins at an arbitrary $z_{\text{hi},0} \in Z_{\text{hi},0}$. Let $\{R_t\}$ be the transformation family that describes the robot's motion when tracking the trajectory parameterized by k. Consider a pair (p_1, p_2) of adjacent points of X_p. Recall that the function `sample` returns the endpoints of any line segment (as in Definition 91) or arc (as in Definition 93), in addition to points spaced along the line segment or arc if necessary. Therefore, by Algorithm 2, (p_1, p_2) is either from a line segment or from an arc of ∂P_b. Recall that, by Lemma 47, ∂P_b consists exclusively of line segments and arcs. By construction, if p_1 is on a line segment (resp. arc), then p_2 is within the distance r (resp. a) along the line segment; this also holds if either point is an endpoint of a line segment or arc.

Consider the case when (p_1, p_2) is from an arbitrary line segment L_i of ∂P_b. By (4.15), the distance from P to any point on L_i is b. By Lemma 44, when tracking the trajectory parameterized by k, the robot can approach infinitesimally close to p_1 and/or p_2, but cannot contain them, for any $t \in [0, T]$. So, by Lemma 98 and continuity of the robot's trajectory, no point in the robot can penetrate farther than b through L_i.

Now consider when (p_1, p_2) is from an arbitrary arc A_i of ∂P_b. By Equation (4.15), the distance from X_{obs} to any point on A_i is b. Each such arc is a section of a circle of radius b. By Lemma 44, the robot cannot contain p_1 or p_2 for any $t \in [0, T]$. So, by Lemma 99 and continuity of the robot's trajectory, the robot cannot pass farther than the distance b into A_i through the chord of A_i with endpoints p_1 and p_2.

Since L_i and A_i were arbitrary, there does not exist any $t \in [0, T]$ for which $R_t X_0 \cap X_{\text{obs}}$ is nonempty. In other words, the robot does not collide with P by passing through any line segment or arc of ∂P_b. Since k was arbitrary, one can conclude that there does not exist any $k \in \pi_K(X_p)^C$ for which the robot collides with any obstacle. Therefore, $\pi_K(X_p)^C \subseteq K_{\text{safe}}$. □

APPENDIX C

Nonlinear Model Predictive Control

This section provides an example of how slack variables are incorporated into the NMPC program (7.35) to ensure that a feasible solution always exists. Positive slack variables s_j are introduced for $j \in \{0, \ldots, n_{\text{pred}}\}$ The cost function (7.32) is modified as

$$J(z, \tilde{u}, s) = \frac{1}{2}\|z - z_{\text{ref}}\|_Q^2 + \frac{1}{2}\|\tilde{u} - \tilde{u}_{\text{ref}}\|_R^2 + r_s s \tag{C.1}$$

where $r_s \in R_{>0}$ penalizes the slack variables in the L1 sense. Take the constraints (7.43) as an example. These are reformulated as

$$y_{\text{min}} - s_j \leq y_j \leq y_{\text{max}} + s_j, \tag{C.2a}$$
$$-\delta_{\text{max}} - s_j \leq \delta_j \leq \delta_{\text{max}} + s_j, \tag{C.2b}$$
$$-\dot{\delta}_{\text{max}} - s_j \leq \dot{\delta}_j \leq \dot{\delta}_{\text{max}} + s_j, \tag{C.2c}$$
$$\omega_{i,\min} - s_j \leq \omega_{j,i} \leq \omega_{i,\max} + s_j,\ i \in \{f, r\}, \tag{C.2d}$$
$$\alpha_{i,\min} - s_j \leq \alpha_{j,i} \leq \alpha_{i,\max} + s_j,\ i \in \{f, r\}, \tag{C.2e}$$
$$s_j \geq 0 \tag{C.2f}$$

for $j \in \{0, \ldots, n_{\text{pred}}\}$.

BIBLIOGRAPHY

[Aco07] Velodyne Acoustics. Velodyne's hdl-64e: A high definition lidar sensor for 3-d applications. *White Paper*, 2007.

[AD14] Matthias Althoff and John M Dolan. Online verification of automated road vehicles using reachability analysis. *IEEE Transactions on Robotics*, 30(4):903–918, 2014.

[AG07] Søren Asmussen and Peter W Glynn. *Stochastic simulation: algorithms and analysis*, volume 57. Springer Science & Business Media, 2007.

[AGH+18a] Joel A. E. Andersson, Joris Gillis, Greg Horn, James B. Rawlings, and Moritz Diehl. Casadi–a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 2018.

[AGH+18b] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, In Press, 2018.

[AM11] Karsten Ahnert and Mario Mulansky. Odeint - Solving ordinary differential equations in C++. *CoRR*, abs/1110.3397, 2011.

[APT12] Changsun Ahn, Huei Peng, and H Eric Tseng. Robust estimation of road friction coefficient using lateral and longitudinal vehicle dynamics. *Vehicle System Dynamics*, 50(6):961–985, 2012.

[ATH+21] Anil Alan, Andrew J Taylor, Chaozhe R He, Gábor Orosz, and Aaron D Ames. Safe controller synthesis with tunable input-to-state safe control barrier functions. *IEEE Control Systems Letters*, 2021.

[AVJR21] Cyrus Anderson, Ram Vasudevan, and Matthew Johnson-Roberson. A kinematic model for trajectory prediction in general highway scenarios. *arXiv preprint arXiv:2103.16673*, 2021.

[AXGT16] Aaron D Ames, Xiangru Xu, Jessy W Grizzle, and Paulo Tabuada. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control*, 62(8):3861–3876, 2016.

[BDC18] Karl Berntorp and Stefano Di Cairano. Tire-stiffness and vehicle-state estimation based on noise-adaptive particle filtering. *IEEE Transactions on Control Systems Technology*, 27(3):1100–1114, 2018.

[Ber20] Karl Berntorp. Online Bayesian tire-friction learning by Gaussian-process state-space models. In *IFAC World Congress*, Berlin, Germany, July 2020.

[Ber21] Karl Berntorp. Bayesian inference and learning of Gaussian-process state-space models. *Automatica*, 2021.

[Bha87] BB Bhattacharyya. One sided chebyshev inequality when the first four moments are known. *Communications in Statistics-Theory and Methods*, 16(9):2789–2791, 1987.

[BHFK+19] Andrea Bajcsy, Sylvia L Herbert, David Fridovich-Keil, Jaime F Fisac, Sampada Deglurkar, Anca D Dragan, and Claire J Tomlin. A scalable framework for real-time multi-robot, multi-human collision avoidance. In *2019 international conference on robotics and automation (ICRA)*, pages 936–943. IEEE, 2019.

[BLW06] Lars Blackmore, Hui Li, and Brian Williams. A probabilistic approach to optimal robust path planning with obstacles. In *2006 American Control Conference*, pages 7–pp. IEEE, 2006.

[BMT12] Andrew J Barry, Anirudha Majumdar, and Russ Tedrake. Safety verification of reactive controllers for uav flight in cluttered environments using barrier certificates. In *2012 IEEE International Conference on Robotics and Automation*, pages 484–490. IEEE, 2012.

[BOLN14] Karl Berntorp, Björn Olofsson, Kristoffer Lundahl, and Lars Nielsen. Models and methodology for optimal trajectory generation in safety-critical road–vehicle manoeuvres. *Vehicle System Dynamics*, 52(10):1304–1332, 2014.

[BOW11] Lars Blackmore, Masahiro Ono, and Brian C Williams. Chance-constrained optimal path planning with obstacles. *IEEE Transactions on Robotics*, 27(6):1080–1094, 2011.

[BQUDC19] Karl Berntorp, Rien Quirynen, Tomoki Uno, and Stefano Di Cairano. Trajectory tracking for autonomous vehicles on varying road surfaces by friction-adaptive nonlinear model predictive control. *Vehicle System Dynamics*, 58(5):705–725, 2019.

[BQV21] Karl Berntorp, Rien Quirynen, and Sean Vaskov. Joint tire-stiffness and vehicle-inertial parameter estimation for improved predictive control. In *American Control Conference, New Orleans, LA*, 2021.

[BWWN18] Julie Stephany Berrio, James Ward, Stewart Worrall, and Eduardo M. Nebot. Identifying robust landmarks in feature-based maps. *CoRR*, abs/1809.09774, 2018. View online.

[BZAF19] Ivo Batkovic, Mario Zanon, Mohammad Ali, and Paolo Falcone. Real-time constrained trajectory planning and vehicle control for proactive autonomous driving with road users. In *2019 18th European Control Conference (ECC)*, pages 256–262. IEEE, 2019.

[BZTB18] Monimoy Bujarbaruah, Xiaojing Zhang, H Eric Tseng, and Francesco Borrelli. Adaptive MPC for autonomous lane keeping. *arXiv preprint arXiv:1806.04335*, 2018.

[Can29] Francesco Paolo Cantelli. Sui confini della probabilita. In *Atti del Congresso Internazionale dei Matematici: Bologna del 3 al 10 de settembre di 1928*, pages 47–60, 1929.

[CBS+21] Margaret P Chapman, Riccardo Bonalli, Kevin M Smith, Insoon Yang, Marco Pavone, and Claire J Tomlin. Risk-sensitive safety analysis using conditional value-at-risk. *arXiv preprint arXiv:2101.12086*, 2021.

[CCS58] Abraham Charnes, William W Cooper, and Gifford H Symonds. Cost horizons and certainty equivalents: an approach to stochastic programming of heating oil. *Management science*, 4(3):235–263, 1958.

[CHT16] Mo Chen, Sylvia Herbert, and Claire J Tomlin. Exact and efficient hamilton-jacobi-based guaranteed safety analysis via system decomposition. *arXiv preprint arXiv:1609.05248*, 2016.

[CHV+17] M. Chen, S. L. Herbert, M. S. Vashishtha, S. Bansal, and C. J. Tomlin. Decomposition of Reachable Sets and Tubes for a Class of Nonlinear Systems. *ArXiv e-prints*, July 2017.

[CLL14] Bo-Chiuan Chen, Bi-Cheng Luan, and Kangwon Lee. Design of lane keeping system using adaptive model predictive control. In *2014 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 922–926. IEEE, 2014.

[CLLSA+20] Manuel Castillo-Lopez, Philippe Ludivig, Seyed Amin Sajadi-Alamdari, Jose Luis Sanchez-Lopez, Miguel A Olivares-Mendez, and Holger Voos. A real-time approach for chance-constrained motion planning with dynamic obstacles. *IEEE Robotics and Automation Letters*, 5(2):3620–3625, 2020.

[COMB19] Richard Cheng, Gábor Orosz, Richard M Murray, and Joel W Burdick. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3387–3395, 2019.

[Cou92] R Craig Coulter. Implementation of the pure pursuit path tracking algorithm. Technical report, Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992.

[CPG17] Yuxiao Chen, Huei Peng, and Jessy W Grizzle. Fast trajectory planning and robust trajectory tracking for pedestrian avoidance. *Ieee Access*, 5:9304–9317, 2017.

[CWHL21] Fabian Christ, Alexander Wischnewski, Alexander Heilmeier, and Boris Lohmann. Time-optimal trajectory planning for a race car considering variable tyre-road friction coefficients. *Vehicle system dynamics*, 59(4):588–612, 2021.

[DAB20] Anushri Dixit, Mohamadreza Ahmadi, and Joel W Burdick. Risk-sensitive motion planning using entropic value-at-risk. *arXiv preprint arXiv:2011.11211*, 2020.

[DCKB16] Stefano Di Cairano, UV Kalabić, and Karl Berntorp. Vehicle tracking control on piecewise-clothoidal trajectories by mpc with guaranteed error bounds. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 709–714. IEEE, 2016.

[DCTBB12] Stefano Di Cairano, Hongtei Eric Tseng, Daniele Bernardini, and Alberto Bemporad. Vehicle yaw stability control by coordinated active front steering and differential braking in the tire sideslip angles domain. *IEEE Transactions on Control Systems Technology*, 21(4):1236–1248, 2012.

[DJ09] Arnaud Doucet and A. M. Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. In D. Crisan and B. Rozovsky, editors, *Handbook of Nonlinear Filtering*. Oxford University Press, 2009.

[DR07] Philip J Davis and Philip Rabinowitz. *Methods of numerical integration*. Courier Corporation, 2007.

[DS16] Christopher M. Dellin and Siddhartha S. Srinivasa. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'16, pages 459–467. AAAI Press, 2016. View online.

[DSSW09] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering route planning algorithms. In *Algorithmics of large and complex networks*, pages 117–139. Springer, 2009.

[DWE20] James Dallas, Yifan Weng, and Tulga Ersal. Combined trajectory planning and tracking for autonomous vehicles on deformable terrains. In *Dynamic Systems and Control Conference*, volume 84270, page V001T15A001. American Society of Mechanical Engineers, 2020.

[ES14] Mohamed Elbanhawi and Milan Simic. Sampling-based robot motion planning: A review. *IEEE Access*, 2:56–77, 2014.

[FBA+07] Paolo Falcone, Francesco Borrelli, Jahan Asgari, Hongtei Eric Tseng, and Davor Hrovat. Predictive active steering control for autonomous vehicle systems. *IEEE Transactions on control systems technology*, 15(3):566–580, 2007.

[FDCQ20] Xuhui Feng, Stefano Di Cairano, and Rien Quirynen. Inexact adjoint-based sqp algorithm for real-time stochastic nonlinear mpc. In *IFAC World Congress*, 2020.

[FGZ+13] J. V. Frasch, A. Gray, M. Zanon, H. J. Ferreau, S. Sager, F. Borrelli, and M. Diehl. An auto-generated nonlinear MPC algorithm for real-time obstacle avoidance of ground vehicles. In *2013 European Control Conference (ECC)*, pages 4136–4141, July 2013.

[FHW12] Efi Fogel, Dan Halperin, and Ron Wein. *Minkowksi Sums and Offset Polygons*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[FJSE20] Huckleberry Febbo, Paramsothy Jayakumar, Jeffrey L Stein, and Tulga Ersal. Nloptcontrol: A modeling language for solving optimal control problems. *arXiv preprint arXiv:2003.00142*, 2020.

[FKBF+20] David Fridovich-Keil, Andrea Bajcsy, Jaime F Fisac, Sylvia L Herbert, Steven Wang, Anca D Dragan, and Claire J Tomlin. Confidence-aware motion prediction for real-time collision avoidance1. *The International Journal of Robotics Research*, 39(2-3):250–265, 2020.

[FLJ+17] Huckleberry Febbo, Jiechao Liu, Paramsothy Jayakumar, Jeffrey L Stein, and Tulga Ersal. Moving obstacle avoidance for large, high-speed autonomous ground vehicles. In *2017 American Control Conference (ACC)*, pages 5568–5573. IEEE, 2017.

[FMM20] Chuchu Fan, Kristina Miller, and Sayan Mitra. Fast and guaranteed safe controller synthesis for nonlinear vehicle models. In *International Conference on Computer Aided Verification*, pages 629–652. Springer, 2020.

[GFX13] Yunlong Gao, Yuan Feng, and Lu Xiong. Vehicle longitudinal velocity estimation with adaptive kalman filter. In *Proceedings of the FISITA 2012 World Automotive Congress*, pages 415–423. Springer, 2013.

[GGC+14] Y. Gao, A. Gray, A. Carvalho, H. E. Tseng, and F. Borrelli. Robust nonlinear predictive control for semiautonomous ground vehicles. In *2014 American Control Conference*, pages 4913–4918, June 2014.

[GGL+12] Andrew Gray, Yiqi Gao, Theresa Lin, J Karl Hedrick, H Eric Tseng, and Francesco Borrelli. Predictive control for agile semi-autonomous ground vehicles using motion primitives. In *2012 American Control Conference (ACC)*, pages 4239–4244. IEEE, 2012.

[Gus97] Fredrik Gustafsson. Slip-based tire-road friction estimation. *Automatica*, 33(6):1087–1099, 1997.

[Gus09] F. Gustafsson. Automotive safety systems. *IEEE Signal Processing magazine*, 26(4):32–47, July 2009.

[Gus10] Fredrik Gustafsson. *Statistical sensor fusion*. Studentlitteratur, 2010.

[GZQ+20a] Sébastien Gros, Mario Zanon, Rien Quirynen, Alberto Bemporad, and Moritz Diehl. From linear to nonlinear mpc: bridging the gap via the real-time iteration. *International Journal of Control*, 93(1):62–80, 2020.

[GZQ+20b] Sébastien Gros, Mario Zanon, Rien Quirynen, Alberto Bemporad, and Moritz Diehl. From linear to nonlinear mpc: bridging the gap via the real-time iteration. *International Journal of Control*, 93(1):62–80, 2020.

[Ham18] Tareq Hamadneh. *Bounding polynomials and rational functions in the tensorial and simplicial bernstein forms*. PhD thesis, Universität Konstanz, 2018.

[HCH+17] Sylvia L Herbert, Mo Chen, SooJean Han, Somil Bansal, Jaime F Fisac, and Claire J Tomlin. Fastrack: A modular framework for fast and guaranteed safe motion planning. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 1517–1522. IEEE, 2017.

[HGK10] Thomas M Howard, Colin J Green, and Alonzo Kelly. Receding horizon model-predictive control for mobile robot navigation of intricate paths. In *Field and Service Robotics*, pages 69–78. Springer, 2010.

[HK07] Thomas M. Howard and Alonzo Kelly. Optimal Rough Terrain Trajectory Generation for Wheeled Mobile Robots. *The International Journal of Robotics Research*, 26(2):141–166, 2007.

[HKRA16] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-Time Loop Closure in 2D LIDAR SLAM. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, 2016.

[HKZ19] Lukas Hewing, Juraj Kabzan, and Melanie N Zeilinger. Cautious model predictive control using gaussian process regression. *IEEE Transactions on Control Systems Technology*, 28(6):2736–2743, 2019.

[HLM+01] Rene Henrion, Pu Li, Andris Möller, Marc C Steinbach, Moritz Wendt, and Günter Wozny. Stochastic optimization for operating chemical processes under uncertainty. In *Online optimization of large scale systems*, pages 457–478. Springer, 2001.

[HY20] Astghik Hakobyan and Insoon Yang. Wasserstein distributionally robust motion planning and control with safety constraints using conditional value-at-risk. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 490–496. IEEE, 2020.

[ISO02] ISO 3888-2: 2002. Passenger cars–test track for a severe lane change manoeuvre–part 2: Obstacle avoidance, 2002.

[Jan18] Mrdjan Jankovic. Robust control barrier functions for constrained stabilization of nonlinear systems. *Automatica*, 96:359–367, 2018.

[JHW21] Ashkan Jasour, Weiqiao Han, and Brian Williams. Convex risk bounded continuous-time trajectory planning in uncertain nonconvex environments. *arXiv preprint arXiv:2106.05489*, 2021.

[JSP18] Lucas Janson, Edward Schmerling, and Marco Pavone. Monte carlo motion planning for robot trajectory optimization under uncertainty. In *Robotics Research*, pages 343–361. Springer, 2018.

[KA17] Markus Koschi and Matthias Althoff. Spot: A tool for set-based prediction of traffic participants. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1686–1693. IEEE, 2017.

[KET17] Seyedmeysam Khaleghian, Anahita Emami, and Saied Taheri. A technical survey on tire-road friction estimation. *Friction*, 5(2):123–146, 2017.

[KHV19] Shreyas Kousik, Patrick Holmes, and Ram Vasudevan. Safe, aggressive quadrotor flight via reachability-based trajectory design. In *ASME 2019 Dynamic Systems and Control Conference*. American Society of Mechanical Engineers Digital Collection, 2019.

[KLS11] Matej Kristan, Aleš Leonardis, and Danijel Skočaj. Multivariate online kernel density estimation with gaussian kernels. *Pattern Recognition*, 44(10-11):2630–2642, 2011.

[KMKR20] Rachel E Keil, Alexander T Miller, Mrinal Kumar, and Anil V Rao. Method for chance constrained optimal control using biased kernel density estimators. *arXiv preprint arXiv:2003.08010*, 2020.

[KPU02] Pavlo Krokhmal, Jonas Palmquist, and Stanislav Uryasev. Portfolio optimization with conditional value-at-risk objective and constraints. *Journal of risk*, 4:43–68, 2002.

[KQCD15] Christos Katrakazas, Mohammed Quddus, Wen-Hua Chen, and Lipika Deka. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, 60:416–442, 2015.

[KTF+09] Yoshiaki Kuwata, Justin Teo, Gaston Fiore, Sertac Karaman, Emilio Frazzoli, and Jonathan P How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on control systems technology*, 17(5):1105–1118, 2009.

[KVB+20] Shreyas Kousik, Sean Vaskov, Fan Bu, Matthew Johnson-Roberson, and Ram Vasudevan. Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots. *The International Journal of Robotics Research*, 39(12):1419–1469, 2020.

[KVJRV17] Shreyas Kousik, Sean Vaskov, Matthew Johnson-Roberson, and Ram Vasudevan. Safe trajectory synthesis for autonomous driving in unforeseen environments. In *Dynamic Systems and Control Conference*, volume 58271, page V001T44A005. American Society of Mechanical Engineers, 2017.

[KZZV20] Shreyas Kousik, Bohao Zhang, Pengcheng Zhao, and Ram Vasudevan. Safe, optimal, real-time trajectory planning with a parallel constrained bernstein algorithm. *arXiv preprint arXiv:2003.01758*, 2020.

[Las09] Jean Bernard Lasserre. Moments, positive polynomials and their applications, 2009.

[LCZW19] Fen Lin, Yuke Chen, Youqun Zhao, and Shaobo Wang. Path tracking of autonomous vehicle based on adaptive model predictive control. *International Journal of Advanced Robotic Systems*, 16(5):1729881419880089, 2019.

[LDM15] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1: 43 scale rc cars. *Optimal Control Applications and Methods*, 36(5):628–647, 2015.

[LKJ01] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

[McN11] Matthew McNaughton. *Parallel Algorithms for Real-time Motion Planning*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 7 2011. .

[Mos10] Mosek ApS. The MOSEK optimization software, 2010.

[MT17] Anirudha Majumdar and Russ Tedrake. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research*, 36(8):947–982, 2017.

[MUDL11] Matthew McNaughton, Chris Urmson, John M Dolan, and Jin-Woo Lee. Motion planning for autonomous driving with a conformal spatiotemporal lattice. In *2011 IEEE International Conference on Robotics and Automation*, pages 4889–4895. IEEE, 2011.

[Mun00] James Munkres. *Topology (2nd Edition)*. Pearson, 2 edition, January 2000.

[MVTT14] Anirudha Majumdar, Ram Vasudevan, Mark M Tobenkin, and Russ Tedrake. Convex optimization of nonlinear feedback controllers via occupation measures. *The International Journal of Robotics Research*, 33(9):1209–1230, 2014.

[NA11] Paluri SV Nataraj and M Arounassalame. Constrained global optimization of multivariate polynomials using bernstein branch and prune algorithm. *Journal of global optimization*, 49(2):185–212, 2011.

[NPDC+21] Truls Nyberg, Christian Pek, Laura Dal Col, Christoffer Norén, and Jana Tumova. Risk-aware motion planning for autonomous vehicles with safety specifications. In *IEEE Intelligent Vehicles Symposium*, 2021.

[NS07] Arkadi Nemirovski and Alexander Shapiro. Convex approximations of chance constrained programs. *SIAM Journal on Optimization*, 17(4):969–996, 2007.

[NW06] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006.

[OW08] Masahiro Ono and Brian C Williams. Iterative risk allocation: A new approach to robust model predictive control with a joint chance constraint. In *2008 47th IEEE Conference on Decision and Control*, pages 3427–3432. IEEE, 2008.

[PA18] Christian Pek and Matthias Althoff. Computationally efficient fail-safe trajectory planning for self-driving vehicles using convex optimization. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 1447–1454. IEEE, 2018.

[PAdNdLF17] Philip Polack, Florent Altché, Brigitte d'Andréa Novel, and Arnaud de La Fortelle. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *2017 IEEE intelligent vehicles symposium (IV)*, pages 812–818. IEEE, 2017.

[Par00] Pablo A Parrilo. Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization, 2000.

[PB00] Aurelio Piazzi and C Guarino Lo Bianco. Quintic g/sup 2/-splines for trajectory planning of autonomous vehicles. In *Proceedings of the IEEE intelligent vehicles symposium 2000 (Cat. No. 00TH8511)*, pages 198–203. IEEE, 2000.

[PH06] B Pacejka Hans. Tire and vehicle dynamics. *Warrendale: SAE*, 2006.

[PKA16] L. Palmieri, S. Koenig, and K. O. Arras. RRT-based nonholonomic motion planning using any-angle path biasing. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2775–2781, May 2016.

[PLM06] R. Pepy, A. Lambert, and H. Mounier. Path Planning using a Dynamic Vehicle Model. In *2006 2nd International Conference on Information Communication Technologies*, volume 1, pages 781–786, 2006.

[PR14] Michael A Patterson and Anil V Rao. Gpops-ii: A matlab software for solving multiple-phase optimal control problems using hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming. *ACM Transactions on Mathematical Software (TOMS)*, 41(1):1–37, 2014.

[Pré13] András Prékopa. *Stochastic programming*, volume 324. Springer Science & Business Media, 2013.

[PWT+07] William H Press, H William, Saul A Teukolsky, William T Vetterling, A Saul, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[PY19] Dávid Papp and Sercan Yildiz. Sum-of-squares optimization without semidefinite programming. *SIAM Journal on Optimization*, 29(1):822–851, 2019.

[QBDC18] Rien Quirynen, Karl Berntorp, and Stefano Di Cairano. Embedded optimization algorithms for steering in autonomous vehicles based on nonlinear model predictive control. In *2018 Annual American Control Conference (ACC)*, pages 3251–3256. IEEE, 2018.

[QDC20] Rien Quirynen and Stefano Di Cairano. PRESAS: Block-structured preconditioning of iterative solvers within a primal active-set method for fast model predictive control. *Optimal Control Applications and Methods*, 2020.

[Raj11] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[RC95] Dietmar Ratz and Tibor Csendes. On the selection of subdivision directions in interval branch-and-bound methods for global optimization. *Journal of Global Optimization*, 7(2):183–207, 1995.

[RKCL16] Yadollah Rasekhipour, Amir Khajepour, Shih-Ken Chen, and Bakhtiar Litkouhi. A potential field-based model predictive path-planning controller for autonomous road vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 18(5):1255–1267, 2016.

[RW06] CE Rasmussen and CKI Williams. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, USA, 2006.

[SHB14] Dieter Schramm, Manfred Hiller, and Roberto Bardini. *Vehicle Dynamics Modeling and Simulation*. Springer, 2014.

[SNR+18] Michael J Sprung, Long X Nguyen, Demi Riley, Siyan Zhou, Alex Lawson, et al. national transportation statistics 2018, 2018.

[SP08] AJ Shaiju and Ian R Petersen. Formulas for discrete time lqr, lqg, leqg and minimax lqg optimal control problems. *IFAC Proceedings Volumes*, 41(2):8773–8778, 2008.

[SSSS17] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. On a formal model of safe and scalable self-driving cars. *arXiv preprint arXiv:1708.06374*, 2017.

[Str82] Gilbert Strang. The Width of a Chair. *The American Mathematical Monthly*, 89(8):529–534, 1982.

[SVBT14] Victor Shia, Ram Vasudevan, Ruzena Bajcsy, and Russ Tedrake. Convex computation of the reachable set for controlled polynomial hybrid systems. In *53rd IEEE Conference on Decision and Control*, pages 1499–1506. IEEE, 2014.

[Sve07] Jacob Svendenius. *Tire modeling and friction estimation*. PhD thesis, Department of Automatic Control, Lund University Lund, Sweden, 2007.

[TFS19] Angelos Toytziaridis, Paolo Falcone, and Jonas Sjöberg. A data-driven markovian framework for multi-agent pedestrian collision risk prediction. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 777–782. IEEE, 2019.

[TG17] Jihad Titi and Jürgen Garloff. Fast determination of the tensorial and simplicial bernstein forms of multivariate polynomials and rational functions. Technical Report 361, Universitat Konstanz, 2017.

[Thr02] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[TPM13] Mark M Tobenkin, Frank Permenter, and Alexandre Megretski. Spotless polynomial and conic optimization, 2013.

[TSC06] Mara Tanelli, Sergio M Savaresi, and Carlo Cantoni. Longitudinal vehicle speed estimation for traction and braking control systems. In *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, pages 2790–2795. IEEE, 2006.

[TVC+15a] D. Telen, M. Vallerio, L. Cabianca, B. Houska, J. Van Impe, and F. Logist. Approximate robust optimization of nonlinear systems under parametric uncertainty and process noise. *J. Proc. Control*, 33:140–154, 2015.

[TVC+15b] Dries Telen, Mattia Vallerio, Lorenzo Cabianca, Boris Houska, Jan Van Impe, and Filip Logist. Approximate robust optimization of nonlinear systems under parametric uncertainty and process noise. *Journal of Process Control*, 33:140–154, 2015.

[UAB+08] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, M. N. Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas M. Howard, Sascha Kolski, Alonzo Kelly, Maxim Likhachev, Matt McNaughton, Nick Miller, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul Rybski, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod Snider, Anthony Stentz, William Whittaker, Ziv Wolkowicki, Jason Ziglar, Hong Bae, Thomas Brown, Daniel Demitrish, Bakhtiar Litkouhi, Jim Nickolaou, Varsha Sadekar, Wende Zhang, Joshua Struble, Michael Taylor, Michael Darms, and Dave Ferguson. Autonomous driving in urban environments: Boss and the Urban Challenge. *Journal of Field Robotics*, 25(8):425–466, Jul 2008.

[VKL+19] Sean Vaskov, Shreyas Kousik, Hannah Larson, Fan Bu, James Ward, Stewart Worrall, Matthew Johnson-Roberson, and Ram Vasudevan. Towards provably not-at-fault control of autonomous robots in arbitrary dynamic environments. *arXiv preprint arXiv:1902.02851*, 2019.

[VLK+19] Sean Vaskov, Hannah Larson, Shreyas Kousik, Matthew Johnson-Roberson, and Ram Vasudevan. Not-at-fault driving in traffic: A reachability-based approach. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2785–2790. IEEE, 2019.

[VQB21] Sean Vaskov, Rien Quirynen, and Karl Berntorp. Cornering stiffness adaptive, stochastic nonlinear model predictive control for vehicles. In *American Control Conference, New Orleans, LA*, 2021.

[VSK+19] Sean Vaskov, Utkarsh Sharma, Shreyas Kousik, Matthew Johnson-Roberson, and Ramanarayan Vasudevan. Guaranteed safe reachability-based trajectory design for

a high-fidelity model of an autonomous passenger vehicle. In *2019 American Control Conference (ACC)*, pages 705–710. IEEE, 2019.

[WDSE20] John Wurts, James Dallas, Jeffrey L Stein, and Tulga Ersal. Adaptive nonlinear model predictive control for collision imminent steering with uncertain coefficient of friction. In *2020 American Control Conference (ACC)*, pages 4856–4861. IEEE, 2020.

[WHJW20] Allen Wang, Xin Huang, Ashkan Jasour, and Brian Williams. Fast risk assessment for autonomous vehicles using learned models of agent futures. *arXiv preprint arXiv:2005.13458*, 2020.

[WJW20] Allen Wang, Ashkan Jasour, and Brian Williams. Non-gaussian chance-constrained trajectory planning for autonomous vehicles in the presence of uncertain agents. *arXiv preprint arXiv:2002.10999*, 2020.

[WSE18] John Wurts, Jeffrey L Stein, and Tulga Ersal. Collision imminent steering using nonlinear model predictive control. In *2018 Annual American Control Conference (ACC)*, pages 4772–4777. IEEE, 2018.

[WSE20] John Wurts, Jeffrey L Stein, and Tulga Ersal. Collision imminent steering at high speed using nonlinear model predictive control. *IEEE Transactions on Vehicular Technology*, 69(8):8278–8289, 2020.

[YVJR19] Ming-Yuan Yu, Ram Vasudevan, and Matthew Johnson-Roberson. Occlusion-aware risk assessment for autonomous driving in urban environments. *IEEE Robotics and Automation Letters*, 4(2):2235–2241, 2019.

[ZG98] Michael Zettler and Jürgen Garloff. Robustness analysis of polynomials with polynomial parameter dependency using Bernstein expansion. *IEEE Transactions on Automatic Control*, 43(3):425–431, 1998.

[Zil96] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, 17(3):73–73, 1996.

[ZK14] Zinan Zhao and Mrinal Kumar. A mcmc/bernstein approach to chance constrained programs. In *2014 American Control Conference*, pages 4318–4323. IEEE, 2014.

[ZK17] Zinan Zhao and Mrinal Kumar. Split-bernstein approach to chance-constrained optimal control. *Journal of Guidance, Control, and Dynamics*, 40(11):2782–2795, 2017.