UNIVERSITY OF CALIFORNIA SAN DIEGO

Learning Safety Certificates for Neural Controllers

A Thesis submitted in partial satisfaction of the requirements
for the degree Master of Science

in

Computer Science

by

Zhizhen Qin

Committee in charge:

Professor Sicun Gao, Chair
Professor Henrik I. Christensen
Professor Sylvia Herbert

2021

The thesis of Zhizhen Qin is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2021

TABLE OF CONTENTS

Thesis Approval Page iii

Table of Contents iv

List of Figures vi

Acknowledgements vii

Vita viii

Abstract of the Thesis ix

Chapter 1 Introduction 1

Chapter 2 Related Work 4
2.1 Safety of Path-Tracking Control for Autonomous Driving 4
2.2 Synthesis and Learning of Control Barrier Functions 5
2.3 Neural network robustness verification 5

Chapter 3 Preliminaries 7
3.1 Kinematic and Dynamic Models of Car-like Vehicles 7
3.2 Barrier Functions 9
3.3 Policy Optimization 10
3.4 Robustness of Neural Networks 11

Chapter 4 Learning Neural Barriers for Path-Tracking 13
4.1 Sampling Safe and Unsafe States 15
4.2 Learning Neural Barrier Functions 15

Chapter 5 Certifying Neural Barrier Functions 17
5.1 Bounding Partial Derivatives of Neural Barriers 18
5.2 Bounding the Dynamics 20
5.3 Retraining Neural Barrier Function 21

Chapter 6 Experiments 23
6.1 Environments 23
6.2 Training the Neural Controller 24
6.3 Evaluating Lie Derivatives in Different Environments 25
6.4 Training the Barrier Function 28
6.5 Consistency of Performance with Learned Barriers 33
6.6 Importance of retraining 33

Chapter 7 Conclusion 35

Bibliography 36

LIST OF FIGURES

Figure 1.1: Left: Control performance comparison among Stanley, MPC and the neural controller. Right: Snapshot of the TORCS environment for high-fidelity simulation of vehicle dynamics. 2

Figure 3.1: Left: kinematic model. Right: dynamic model 8

Figure 4.1: Overall flowchart for learning and certifying neural barrier functions. . . . 13

Figure 6.1: Evaluation environments. Left:A path for the kinematic and dynamic model simulation environments. Right: Path in the TORCS vehicle dynamics simulation environment. 23

Figure 6.2: Trajectories of neural controller on dynamic model. Left: angle error, distance error and longitudinal speed. Right: angle error, distance error and lateral speed 26

Figure 6.3: Vector field of the system dynamics calculated with nearest neighbor approach for the dynamic model. 26

Figure 6.4: Full-state control barrier function of kinematic model with target speed of $30m/s$. The points show barrier function boundary, and the red points represent the region with lie derivative violations. The violation rate on the boundary is 0.95% 28

Figure 6.5: Barrier function of dynamic model and TORCS environment, certified on partial observation with distance and angle errors. The blue points are the barrier, and yellow points are the collected (safe) observations. 29

Figure 6.6: 2D barriers of dynamic model and TORCS environment. The surface plot shows the value of learned barrier function, and the x-y plane shows the level sets, with vehicle trajectories within the safe set. The zero-level set is colored in black. 30

Figure 6.7: Illustration of the barrier on TORCS model: when the vehicle is approaching the boundary, the neural controller applies brake and steers the vehicle to the side where the barrier function value decreases. 31

Figure 6.8: Performance comparison among Stanley controller, MPC and neural controller in path tracking tasks. 32

Figure 6.9: Illustration of continuous training. Left: initial barrier (blue and red) and original safe data (green); Right: final barrier after continuous training and relabeled violations (safe: light green; unsafe: yellow). 33

ACKNOWLEDGEMENTS

I would like express my sincere gratitude to Professor Sicun Gao for his support as the chair of my committee and my research advisor. I thank him for providing me an opportunity to do research in the field of robotics control, reinforcement learning and artificial intelligence. His guidance has proved to be invaluable and helped me all the time of research and writing of this thesis.

I would like to thank Prof. Tsui-Wei (Lily) Weng for her expertise and assistance throughout our project. Her encouragement and novel ideas greatly helped me navigate through and improve my research. I offer my sincere appreciation for the learning opportunity provided by Prof. Weng.

Besides my advisor, I would like to thank the rest of my committee members: Prof. Henrik I. Christensen and Prof. Sylvia Herbert, for their effort to review my research and their valuable comments.

The thesis is currently being prepared for submission for publication of the material. It is a joint work with Tsui-Wei (Lily) Weng and Sicun Gao. The thesis author was the primary author of this material.

VITA

2017-2018	Undergraduate Tutor, University of California San Diego
2018	B. S. in Computer Engineering, University of California San Diego
2020-2021	Graduate Teaching Assistant, University of California San Diego
2021	M. S. in Computer Science, University of California San Diego

ABSTRACT OF THE THESIS

Learning Safety Certificates for Neural Controllers

by

Zhizhen Qin

Master of Science in Computer Science

University of California San Diego, 2021

Professor Sicun Gao, Chair

Learning-based methods are promising for tackling the inherent nonlinearity and model uncertainty in path-tracking control problems, but the lack of safety guarantee of neural controllers restricts their practical use in self-driving vehicles. We propose methods for training and certifying barrier functions that are themselves represented as neural networks, for ensuring safety properties of learning-based neural controllers in self-driving with realistic nonlinear dynamics. We describe how to identify safe and unsafe regions of the state space and minimize the violation of the barrier conditions to train neural barrier functions. We then show how to leverage the recent advances in robustness analysis of neural networks to bound the Lie derivatives of the barrier functions and certify the barrier conditions. Although understanding the vehicle dynamics

is important for designing the training and certification procedures, the proposed algorithms are sampling-based and do not assume access to analytic forms of the dynamics. Thus the methods are generally applicable to nonlinear vehicle dynamics with model uncertainty and partial observability. We demonstrate the effectiveness of the methods for quantitatively certifying safety of neural controllers in different simulation environments ranging from simple kinematic models to the TORCS high-fidelity vehicle dynamics modeling environment as a blackbox simulator.

Chapter 1

Introduction

Safe motion control for path-tracking is crucial for reliable autonomous driving. Widely-adopted control methods [1, 2] have inherent difficulty with nonlinearity and uncertainty that cannot be ignored when the vehicles are operating at relatively high speed or under adverse road conditions. As a result, many commercial self-driving systems exhibit unsafe motion control that led to serious accidents. This challenge of nonlinear control is actively studied in ongoing research, with best performance typically obtained through Model-Predictive Control (MPC) schemes [2, 3], which requires high-frequency online optimization, precise dynamic modeling, and good initialization through human intervention (parallel autonomy) [4]. However, the reliance on online computation in MPC-based approaches makes it very difficult to provide general safety guarantees without drastic simplification of the control problems. The question of whether there exist direct feedback control laws for path-tracking with strong safety guarantees for realistic vehicle dynamics remains largely open.

Learning-based methods have shown great promise as the new framework for nonlinear control design. With appropriate training, control laws represented by neural network functions can often show better performance on highly nonlinear systems with model uncertainty. For instance, in Figure 1.1 we show the control performance of the standard Stanley control method [5],

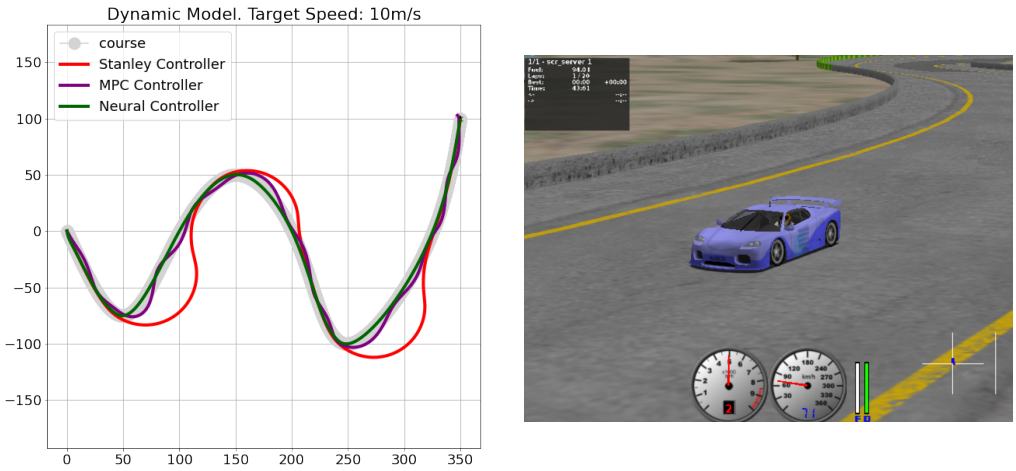


Figure 1.1: Left: Control performance comparison among Stanley, MPC and the neural controller. Right: Snapshot of the TORCS environment for high-fidelity simulation of vehicle dynamics.

Model Predictive Control (MPC) [2, 3] and and that of a neural network controller trained by standard policy optimization methods [6, 7]. The environment simulates vehicle dynamics with inertia effects and lateral slip on a curvy road. Although the Stanley controller is exponentially stable under simple vehicle dynamics, its performance is clearly unsafe in this case. In comparison, the neural controller delivers impressive tracking performance. However, the widely-held concern about the neural controller is in its use of highly nonlinear functions that can unlikely be analyzed and understood as well as the Stanley controller. How can we become confident that neural control policies can indeed deliver safe path tracking in realistic self-driving vehicles?

In this paper, we propose the first systematic approach towards obtaining safety guarantees for neural control methods for path-tracking under realistic vehicle dynamics. We develop learning-based methods for synthesizing and certifying neural barrier functions for the neurally-controlled dynamics of the vehicle. The safety property of the system is established by showing that the neurally-controlled dynamics can not escape the learned barrier and reach unsafe states that correspond to drifting off the paths. To this end, we leverage recent advances in robustness certification of neural networks [8, 9]. The core of the certification procedures is to rigorously bound the Lie derivatives of the learned neural barrier function around sampled states along the

flow of the vehicle dynamics. With these methods, we are able to certify the safety boundary of neural controllers for over 99% of the barrier in the simple kinetic model, 86% for the dynamic model with inertial effects and lateral slip, and 91% in the TORCS racing environment. Moreover, we can completely visualize the sparse regions where the barrier conditions fail the certification, and such regions can be used as monitoring mechanisms to safeguard the use of the neural policies away from potentially unsafe circumstances.

We demonstrate the full pipeline of synthesizing and certifying neural control policies for autonomous path tracking with guaranteed safety margin, by learning neural barriers. Overall we make the following contributions:

- We learn neural barrier functions for learning-based neural controllers for path-tracking with either full-state or partial-state observably (Chapter 4). We show how to identify safe and unsafe regions of the state space and how to learn neural barrier functions with access to only partial state information.

- We show how to certify neural barrier functions by rigorously analyzing the interval bounds on the Lie derivatives of neural barrier functions (Chapter 5). We analyze the numerical properties of the barrier functions with sampling-based estimate of the system dynamics and leverage recent advances in robustness analysis of neural networks.

- We evaluate the proposed methods (Chapter 6) using the following three types of simulation environments of increasing difficulty: kinematic low-speed model with arbitrary sampling access, dynamic model with lateral slip and sampling access, and the blackbox simulator TORCS with high-fidelity vehicle dynamics and limited sampling access. We give quantitative certification results for the neural controllers in all three cases.

To the best of our knowledge, the proposed approach is the first demonstration of the feasibility of quantitative certification of the safety of neural controllers for path-tracking control for autonomous driving under realistic vehicle dynamics.

Chapter 2

Related Work

2.1 Safety of Path-Tracking Control for Autonomous Driving

Certification of safety of path tracking control has been widely studied through reachability analysis [10, 11, 12, 13, 14, 15, 16, 17]. Such methods guarantee safety by certifying reach-avoid sets. Reachability analysis methods are computationally intensive especially for nonlinear dynamical systems. Many approaches have thus focused on reducing the practical computational complexity for using these methods. For instance, [11, 12, 13, 14, 15] partitioning the region into smaller regions with simpler dynamics or finding dynamics within the reachable set. [16, 17] reformulates Hamilton-Jacobi equations to make the computational complexity only be exponential to state variables. [18] uses simple dynamic models to roughly bound tracking errors first, and then fine-tune the planning behavior close to obstacles to guarantee overall safety. barrier function-based methods have major advantages because of their inductive nature for making infinite-horizon guarantees and that the barrier conditions only need to be checked around the safety boundaries rather than the entire state space. The Stanley controller is a standard and well-studied feedback control law for path-tracking in self-driving cars, after Stanford's entry of the same name won the DARPA Grand Challenge 2005 [5, 19, 1, 20, 21]. Although the Stanley

controller has exponential stability properties under low speed and tire no-slip assumption for the kinematic bicycle model, its performance degrades in more challenging scenarios[5]. This challenge of nonlinear control is actively studied in ongoing research, with best performance typically obtained through MPC [2, 3], which typically requires high-frequency online optimization, precise dynamic modeling, and good initialization through human intervention [4].

2.2 Synthesis and Learning of Control Barrier Functions

Barrier certificates [22] and control barrier functions [23, 24] were proposed for certifying the safety of controlled dynamical systems by guaranteeing that all states of a system are contained within a forward invariant set. Sum-of-squares programming [25] and quadratic programming [26] are typical computational approaches for synthesizing polynomial barrier functions for polynomial dynamical systems. Synthesizing barrier functions for general nonlinear systems is a well-known challenge, with various analytic and optimized-based approaches for systems with known dynamics [27, 28, 29]. The work in [30] uses control barrier functions to ensure safety of learned control policies. The work in [31] proves stability properties throughout learning by taking advantage of the robustness of control-theoretic priors. Moreover, our methods allows the verification of barrier functions with approximation of the dynamics within small bounded errors, and do not require accurate modeling. This feature is important for quantitative certification of barrier functions with only blackbox access to high-fidelity dynamics simulation.

2.3 Neural network robustness verification

As deep neural networks become prevalent in machine learning and achieve the best performance in many standard benchmarks, their unexpected vulnerability to adversarial examples has raised serious concerns in the machine learning community and thus motivating

the need of formally quantifying the level of vulnerability of well-trained DNN models. As an example, for image classification tasks, the primary goal of robustness quantification tools is to provide a robustness certificate of a given data sample x and a trained DNN such that the top-1 prediction is consistent for perturbation with magnitude less than the robustness certificate. A common model is an ℓ_p-norm bounded perturbation to x, where p usually takes the value $p \in \{1, 2, \infty\}$, approximating the similarity measure of visual perception between x and its perturbed version. Nevertheless, computing the maximum (exact) robustness certificate is computationally intractable for DNNs [32]; hence, recent verification methods based on local Lipschitz constant and convex/linear relaxation have been proposed to compute a non-trivial lower bound of the maximum robustness certificate [33, 34, 35, 8, 36, 37, 38, 39, 40, 41, 9]. In this work, we primarily adapt the efficient linear bounding framework in [8, 36, 40] to certify the barrier functions.

Chapter 3

Preliminaries

3.1 Kinematic and Dynamic Models of Car-like Vehicles

We consider two types of models: a simple kinematic bicycle model with tire no-slip assumption, and a dynamic model that considers slipping of tires in lateral direction. The kinematic bicycle model gives us the ability to learn a barrier function on full-state information, where we can certify safety over the whole space. The dynamic model contains more dimensions and is more accurate compared to the real world, where we were able to train and verify a barrier function over partial observation.

Kinematic Bicycle Model. A standard model for vehicle motion is the kinematic bicycle model in Figure 3.1. This model does not consider inertial effects or lateral slip and is only usable at low-speed. The x, y coordinates and vehicle yaw angle θ are the state variables with the following dynamics:

$$\dot{x} = v\cos(\theta), \dot{y} = v\sin(\theta), \dot{\theta} = \frac{v\tan(\delta)}{L}, \dot{v} = a$$

where the acceleration a and steering input δ are actions given to the controller, and L is the wheel base of the vehicle. From a given x, y and θ, it is straightforward to compute distance error d_e and angle error θ_e, which we will use as the state variables for the neural control policies.

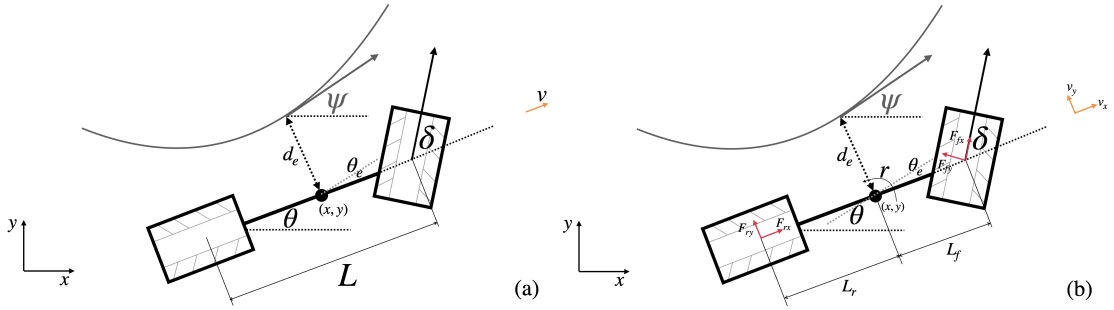


Figure 3.1: Left: kinematic model. Right: dynamic model

Dynamic Model with Inertial Effect and Lateral Slip. When the speed is high, the no-slip assumption of wheels are no longer valid. In this case, a more realistic model uses vehicle dynamics to model the lateral control. The velocities of the vehicle is modelled by the following equations:

$$\dot{x} = v_x \cos(\theta) - v_y \sin(\theta), \dot{y} = v_x \sin(\theta) + v_y \cos(\theta), \dot{\theta} = r$$

The acceleration variables are determined by

$$\begin{aligned}
\dot{v}_x &= a - \frac{1}{m} F_{fy} \sin(\delta) + v_y r \\
\dot{v}_y &= \frac{1}{m}\Big(F_{fy}\cos(\delta) + F_{fr}\Big) - v_x r \\
\dot{r} &= \frac{1}{I_z}\Big(l_f F_{fy}\cos(\delta) - l_r F_{ry}\Big)
\end{aligned}$$

where δ is the steering input; v_x, v_y and r are longitudinal speed, lateral speed and yaw rate, respectively. l_f and l_r are distances from the center of gravity to the front and rear axles. m and I_z are the mass and yaw inertia of the vehicle, respectively. Applying linear approximation for tire cornering stiffness assumption, the lateral forces are

$$F_{i,y} = -c_i \alpha_i$$

where $i \in \{f, r\}$, c_f and c_r are coefficients for the linear approximation of cornering stiffness

parameters of the front and rear tires. α_f and α_r are tire slip angles, with

$$\begin{aligned} \alpha_f &= \left[\tan^{-1}\left(\frac{v_y + l_f r}{v_x}\right) - \delta\right] \\ \alpha_r &= \left[\tan^{-1}\left(\frac{v_y - l_r r}{v_x}\right)\right] \end{aligned} \tag{3.1}$$

Similar to the kinematic model, the distance error and angle error can be derived from x, y, θ and the path geometry.

3.2 Barrier Functions

We express an n-dimensional controlled dynamical system as

$$\dot{x}(t) = f(x(t), u(t)),\ u(t) = g(x(t)),\ x(0) = x_0 \tag{3.2}$$

where $f : D \times \mathbb{R}^m \to \mathbb{R}^n$ is a Lipschitz-continuous vector field, $g : D \to \mathbb{R}^m$ is a control function, and $D \subseteq \mathbb{R}^n$ with $0 \in D$ defines the state space of the system. Each $x(t) \in D$ is called a state vector and $u(t) \in \mathbb{R}^m$ is a control vector.

Definition 1 (Lie Derivatives)**.** Consider the system in (3.2) and let $B : D \to \mathbb{R}$ be a continuously differentiable function. The Lie derivative of B over f is defined as

$$\mathsf{L}_f B(x) = \sum_{i=1}^{n} \frac{\partial B}{\partial x_i} \frac{\mathrm{d}x_i}{\mathrm{d}t} = \sum_{i=1}^{n} \frac{\partial B}{\partial x_i} f_i(t) \tag{3.3}$$

It measures the rate of change of B over time along the direction of the system dynamics of $x(t)$. When the vector field is clear in the context, we also use $\dot{B}(x)$ as a shorthand of $\mathsf{L}_f B(x)$.

Safety properties are formally defined as forward invariance requirement, in the sense that the system dynamics should be contained within safe sets that are disjoint from the unsafe states. Formally we have the following definitions.

Definition 2 (Forward Invariant Sets)**.** Consider the dynamical system f with domain D in Equation 3.2. We say $I \subseteq D$ is a forward invariant set for f if for any $x(0) \in I$ and any $t \geq 0$, we have $x(t) \in I$, i.e. any trajectory that starts in I stays in I forever.

Definition 3 (Barrier Functions [22, 27])**.** Consider a dynamical system $\dot{x} = f$ in state space $X \subseteq \mathbb{R}^n$ with safe region X_s and unsafe region X_u. We call a continuous scalar function $B(x) : \mathbb{R}^n \to \mathbb{R}$ a *barrier function* if:

$$\begin{aligned} B(x) \leq 0, \forall x \in X_s \text{ and } B(x) > 0, \forall x \in X_u \\ \mathsf{L}_f B(x) < 0, \forall x \text{ such that } B(x) = 0 \end{aligned} \tag{3.4}$$

The third condition can be replaced and robustified [24] by

$$\mathsf{L}_f B(x) < -\gamma B(x), \forall x \in X$$

for some positive parameter γ. Note that this condition reduces to the standard requirement on the Lie derivatives at the barrier boundary $(B(x) = 0)$ and poses a slightly stronger requirement at other places. The benefit of using this condition is that it avoids solving the equation $B(x) = 0$ and can be easily turned into a loss function for learning-based approaches.

Because of the third condition $\mathsf{L}_f B(x) < 0, \forall x$ such that $B(x) = 0$, all trajectories starting inside of the barrier would not cross out of the barrier, thus staying in the safe set forever, which implies invariant set.

3.3 Policy Optimization

We will use the following standard notations for reinforcement learning, in which a learning agent interacts with an environment modelled as a Markov Decision Process (MDP) with state space $\mathcal{S}$, action space $\mathcal{A}$ and a distribution $P\colon \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0,1]$. $P(s'|s,a)$ is the probability of transitioning into state s' from state s after taking action a. A reward function

$r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ defines the reward for taking action a in state s and transitioning into s'. Let π_θ denote a stochastic policy $\pi_\theta \colon \mathcal{S} \times \mathcal{A} \to [0,1]$, parameterized by θ. When running a policy $\pi_\theta(a|s)$, we observe a trajectory τ, which includes states $s_t \in \mathcal{S}$, actions $a_t \in \mathcal{A}$ selected according to the policy, and rewards $r_t \in \mathbb{R}$ for every time step t. The goal of the agent is to maximize the expected discounted cumulative return $J(\theta) = \mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1})\right]$. Policy optimization methods [42, 43, 44, 45] estimate policy gradient and use stochastic gradient ascent algorithms to directly improve policy performance. Policy gradient methods [42, 43] use an estimator of policy gradient to improve the policy. A standard gradient estimator is $\hat{g} = \mathbb{E}\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t$ where π_θ is a stochastic policy and $\hat{A}_t$ is an estimator of the advantage function. The expectation is estimated by the empirical average over finite batch of samples [46].

3.4 Robustness of Neural Networks

Given a neural network function $h(z) : \mathbb{R}^d \to \mathbb{R}^n$ and a data sample z_0, our goal is to quantify how the output $h(z)$ changes when the input z_0 is perturbed with some perturbation Δ that lies within an ℓ_p ball with radius ε, i.e. $\|\Delta\|_p \leq \varepsilon$. In other words, we aim at finding an output lower bound $h_L(x)$ and upper bound $h_U(z)$ of $h(z)$ such that

$$h_L(z) \leq h(z) \leq h_U(z),\ \forall \|z - z_0\| \leq \varepsilon.$$

In the linear bounding framework proposed in [8, 36, 40], the authors derived $h_U(z)$ and $h_L(z)$ to be a linear function of z by applying linear upper/lower bounds on each neuron's activation (e.g. ReLU, tanh, sigmoid, etc):

$$h_L(z) = A_L z + B_L,\ h_U(z) = A_U z + B_U,$$

where the linear coefficients and constants are a function of z_0, ϵ, parameters of linear bounds as well as the parameters of the neural network h.

Chapter 4

Learning Neural Barriers for Path-Tracking

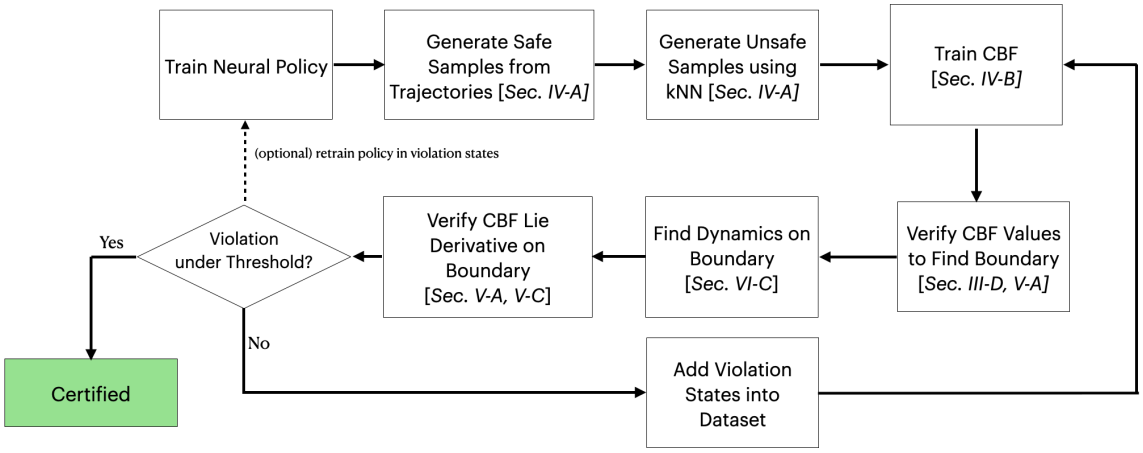


Figure 4.1: Overall flowchart for learning and certifying neural barrier functions.

In this chapter and the next, we describe the detailed procedures for learning and certifying neural barrier functions. The overall pipeline is shown in Figure 4.1 and we will go through each component in the sections below.

We use standard policy optimization algorithms (Proximal Policy Optimization [6] and Soft Actor Critic [7]) to learn a control policy $g(\cdot) : D \to \mathbb{R}^m$. Although a wide range of

regularization techniques can be used to ensure safe learning, our goal is to develop techniques for generic learning-based methods for neural control, and thus did not use special reward engineering. For path tracking, we use a natural design of reward – penalizing the distance error and angular error with the nominal path. When the training of neural policy converges, our main task is to learn a barrier function that can indeed separate the safe and unsafe regions of the trained neural policy (controller). The learning of the neural barrier has the following major steps: generating safe and unsafe samples, sampling-based estimation of Lie derivatives under partial observability, and training the barrier function. The overall algorithm is given in Algorithm 1 and we will describe the details of each step in this chapter.

Algorithm 1 Learning Neural Barrier Functions

Input: Control policy π, unsafe set size N_u, number of nearest neighbors k, number of epochs K, barrier function learning rate α, weight parameters for barrier function loss function w_s, w_u, w_l, γ
Run the neural control policy to collect N_s safe samples X_s, together with the dynamics on these safe samples, which are the differences between adjacent states x' and x for each $x \in X_s$
Initialize unsafe set $X_u = \varnothing$
while $|X_u| < N_u$ **do**
    Sample a batch of candidate observations X_c with size M uniformly randomly from the space
    Initialize to-be-removed observation set X_r
    for every $x_c \in X_c$ **do**
        Neighbors $X_n = k\text{NN}(x_c, X_s \cup X_u \cup X_c)$
        if $|\{x \in X_n | x \in X_s\}| > k/2$ **then**
            $X_r \leftarrow X_r \cup \{x_c\}$
        end if
    end for
    $X_u \leftarrow X_u \cup \left(X_c \setminus X_r\right)$
end while
Initialize barrier function network B_θ
for episodes $= 1, \ldots, K$ **do**
    Sample mini-batches of size n from $X_s \cup X_u$
    $\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$
end for

4.1 Sampling Safe and Unsafe States

We identify the set of safe states X_s and unsafe states X_u as follows.

First, to collect the set X_s of safe state samples, we run the trained control policy for path following from randomized initial positions that are around the path center. Assuming that the control policy is well trained, the vehicle should be able to finish laps without going off the track, and we add all the states that are visited in the sampled trajectory into X_s.

Second, after collecting a number of safe state samples, we generate unsafe state samples $x_u \in X_u$ with k-Nearest Neighbor (kNN) approach. Suppose we want N_u unsafe state samples, at each iteration of generation process, we uniformly sample M observations in the state space and add them into a temporary candidate set X_c. Then, for each candidate observation $x_c \in X_c$, we find its k nearest neighbors inside $X_s \cup X_u \cup X_c$. If more than $k/2$ neighbors are from the safe set, X_s, we remove x_c from X_c. After going through all candidate observations, we add remaining candidates from X_c into unsafe set X_u. We repeat this procedure until we have N_u samples in X_u.

4.2 Learning Neural Barrier Functions

After labeling the safe and unsafe states sets, we train the barrier function using the following loss function, which is similar to the proposals in the recent work [47].

$$\begin{aligned} L(\theta) = w_s \frac{1}{N_s} \sum_{i=0}^{N_s} \text{ReLU}(B(x_s^i)) + w_u \frac{1}{N_u} \sum_{i=0}^{N_u} \text{ReLU}(-B(x_u^i)) \\ + w_l \frac{1}{N_s} \sum_{i=0}^{N_s} \text{ReLU}(\mathsf{L}_f B(x_s^i) + \gamma B(x_s^i))), \end{aligned} \tag{4.1}$$

where w_s, w_u, w_l, γ are hyper parameters for balancing the weights of the different components of the loss. The design of such loss function is to encourage $B(x)$ to satisfy the three conditions of barrier functions in Definition 3: The first term of the loss function pushes the learned barrier function to output positve values on the safe set; the second term pushes it to output negative for

the unsafe set, and the third term governs the lie derivative to be negative for samples around the barrier. Intuitively, $L(\theta)$ is non-negative, and when $L(\theta) = 0$ we have a barrier function that fully satisfies the conditions in Definition 3. Note that the Lie derivative requirement in the third term only operates on the safe set, since we do not have such constraints outside of the barrier.

For generality of the proposed methods, we do not assume analytic knowledge of the system dynamics f. Instead, we approximate the Lie derivative $\mathsf{L}_f V$ along sampled trajectories of the system through $\mathsf{L}_{f,\Delta t} B(s) = \frac{1}{\Delta t}\Big(B(s') - B(s)\Big)$, where s and s' are two consecutive states and Δt is the time difference between them and it is clear that $\lim_{\Delta t \to 0} \mathsf{L}_{f,\Delta t} V(s) = \mathsf{L}_f V(s)$. In practice, the approximation error due to this discretization is taken into account in the certification step, which can be bounded by analyzing the underlying vehicle dynamics.

In the experiments, we trained barrier function with both full state information and partial state observation. We further justified why a barrier function on partial observation is valid in Section 6.3. Also note that the loss function only uses lie derivative on safe samples, which is simultaneously collected as adjacent states x and x' in the trajectories.

Chapter 5

Certifying Neural Barrier Functions

In the previous chapter, we discussed how to learn a neural barrier function by minimizing the barrier loss in Equation 4.1 over the training samples. However, the learned neural barrier function does not guarantee that the barrier conditions of Equation 3.4 hold until it is *certified*. In this chapter, we propose to certify the neural barrier function along with the neural controller based on recent neural network certification tools. The central idea is to analyze whether the barrier conditions are satisfied within local neighborhoods for the state space. In the ideal case, we may be able to certify the entire barrier function for the neural controller, but even if we can not obtain complete certification, the coverage of validated neighborhoods gives us an explicit measurement of the valid regions of the neural barrier. This metric can be used to evaluate the safety of the neural controller, and the validated regions can be turned into a monitor to safeguard the use of the neural policy in real-time. The overall algorithm is given in Algorithm 2 and we will describe the details of each step in this chapter.

To certify a learned neural barrier function, we need to ensure the barrier conditions in Equation 3.4 are all true. The first two conditions involve only the value of the learned barrier function, and are easy to verify. The third condition involves the Lie derivative of the samples on the boundary, which is much harder to ensure and is the main focus of this chapter.

Algorithm 2 Certifying Neural Barrier Functions

Input: Control barrier function B
Initialize certified safe set X_{cs}, unsafe set X_{cu}, boundary set that satisfies lie derivative requirement X_{cb}, and boundary set that violates the requirement X_{vb}
for each point x in the grid **do**
    Calculate lower and upper bounds of barrier function $B_L(x)$ and $B_U(x)$ within an ε box using section 5.1
    if $B_U(x) < 0$ **then** $X_{cs} \leftarrow X_{cs} \cup \{x\}$ ▷ safe
    else if $B_L(x) > 0$ **then** $X_{cu} \leftarrow X_{cu} \cup \{x\}$ ▷ unsafe
    else
        Calculate lower and upper bounds of barrier function derivatives $d_L \leftarrow (\frac{\partial B(x)}{\partial x})_L$ and $d_U \leftarrow (\frac{\partial B(x)}{\partial x})_U$ using Section 5.1
        Find the dynamics of state x using Section 6.3, and calculate lower and upper bounds of dynamics $f_L(x)$ and $f_U(x)$ using Section 5.2 (abbreviated below as f_L, f_U)
        Calculate lie derivative upper bound $\left[\mathsf{L}_f B(x)\right]_U$, which is initialized as $l \leftarrow 0$
        for each dimension i of the observation space **do**
            $l \leftarrow l + \max\left[d_{U,i} f_{U,i},\ d_{U,i} f_{L,i},\ d_{L,i} f_{U,i},\ d_{U,i} f_{U,i}\right]$
        end for
        if $l < 0$ **then**
            $X_{cb} \leftarrow X_{cb} \cup \{x\}$ ▷ certified boundary
        else
            $X_{vb} \leftarrow X_{vb} \cup \{x\}$ ▷ boundary violation
        end if
    end if
end for

5.1 Bounding Partial Derivatives of Neural Barriers

Our goal for certifying the Lie derivative condition is to find all states that are close to the boundary ($B(x) = 0$) and ensure that $\mathsf{L}_f B(x) < 0$ is true for all these states. This is a stronger condition than the original condition, but it is necessary because we can not expect to solve the zero-crossing problem precisely and need to aim for certifying the Lie derivative conditions in a neighborhood around the boundary.

To achieve this, we grid the whole state space and compute the interval bounds (B_L, B_U) of $B(x)$ at the center of each cell. We only need to consider the cells with $B_L < 0$ and $B_U > 0$, which means that they are zero-crossing. For such cells, we could set ε as half cell width and

verify if $\mathsf{L}_f B(x) < 0$ for the cells. If the upper bound of the Lie derivative is negative, then the barrier condition is satisfied for the full local neighborhood $\|x-s\|_\infty \leq \varepsilon$. Recall that the Lie derivative is a dot product between the partial derivatives of B over the state variables and their dynamics:

$$\mathsf{L}_f B(x) := \sum_i \frac{\partial B(x)}{\partial x_i} \frac{dx_i}{dt}$$

The main difficulty is to bound the partial derivative of the neural barrier function $\frac{\partial B(x)}{\partial x_i}$. This can be achieved by using Holder's inequality at each layer. Below, we describe the technique on a barrier function $B(x)$ parameterized by 1 hidden layer NN with tanh activation, denoted as

$$B(x) = W_{1,:}^{(2)} \sigma(W^{(1)} x + b^{(1)}) + b_1^{(2)}.$$

Through differentiation, we know the partial derivative of the neural barrier function is:

$$\frac{\partial B(x)}{\partial x_i} = W_{1,:}^{(2)} \left[\sigma'(W^{(1)} x + b^{(1)}) \odot W_{:,i}^{(1)} \right], \tag{5.1}$$

where $\odot$ is the element-wise multiplication and $\sigma'(\cdot)$ is the derivative of tanh function.

Given a state s and its local neighborhood $\|x-s\|_\infty \leq \varepsilon$, we write the interval bounds of $\frac{\partial B(x)}{\partial x_i}$ as $(l_i^{(2)}, u_i^{(2)})$. Namely,

$$l_i^{(2)} \leq \frac{\partial B(x)}{\partial x_i} \leq u_i^{(2)}, \text{where} \|x-s\|_\infty \leq \varepsilon.$$

We can then derive $u_i^{(2)}$ and $l_i^{(2)}$ via Holder's inequality:

$$\begin{aligned} u_i^{(2)} = \sum_k \left[W_{1,k}^{(2)} W_{k,i}^{(1)} \right]^+ \max\{\sigma'(l_k^{(1)}), \sigma'(u_k^{(1)}), \sigma'(0)\} + \\ + \left[W_{1,k}^{(2)} W_{k,i}^{(1)} \right]^- \min\{\sigma'(l_k^{(1)}), \sigma'(u_k^{(1)})\}, \end{aligned}$$

$$l_i^{(2)} = \sum_k \left[W_{1,k}^{(2)} W_{k,i}^{(1)}\right]^+ \min\{\sigma'(l_k^{(1)}), \sigma'(u_k^{(1)})\} + \\ + \left[W_{1,k}^{(2)} W_{k,i}^{(1)}\right]^- \max\{\sigma'(l_k^{(1)}), \sigma'(u_k^{(1)}), \sigma'(0)\},$$

where $u_k^{(1)} = \varepsilon||W_{k,:}^{(1)}||_1 + W_{k,:}^{(1)} s + b_k^{(1)}$ and $l_k^{(1)} = -\varepsilon||W_{k,:}^{(1)}||_1 + W_{k,:}^{(1)} s + b_k^{(1)}$. Thus, we can propagate the interval bounds on the input x through these equations and obtain bounds on the partial derivatives of $\partial B/\partial x$. The bounds for NN of more layers can be computed by applying this method recursively on each layer.

5.2 Bounding the Dynamics

To find the interval bounds on the Lie derivatives, we need to bound the variation of the dynamics of the system in the neighborhood of a sampled state. Let a be a sampled state, and we write the hyperbox around a as $[a]_\varepsilon$.

First, we estimate the value of $f(a)$ through finite difference over time:

$$f(a) \approx \left.\frac{x(t+\Delta t) - x(t)}{\Delta t}\right|_{x(t)=a}$$

where we assume $\ddot{x}(t+\lambda\Delta t)\Delta t$ is tiny, $\lambda \in [0,1]$. Then consider an arbitrary state $a+p$ in the neighborhood of a:

$$f(a+p) = f(a) + J_f(a)p + \frac{1}{2} p^T H_f(a+\lambda p) p$$

where J_f and H_f are the Jacobian and Hessian-tensor of f (f is vector-valued). We use finite differences to estimate the Jacobian. Writing $f = (f_1, ..., f_n)$, with $\varepsilon \to 0$, we have

$$\left.\frac{\partial f_j}{\partial x_i}\right|_{x=a} \approx \frac{f_j(a+\varepsilon e_i) - f_j(a)}{\varepsilon}$$

where e_i is the unit vector in the x_i coordinate. Thus by taking n (the number of dimensions)

samples of $f(x+\varepsilon e_i)$, we can get the estimate of the Jacobian of f, $J_f(a)$ (each sample used to estimate all functions in f), where e_i is the unit vector in the x_i coordinate, and again assuming that the second-order deviations are negligible by slight bloating of the estimated range. Note that each dimension of sample $f(x)$ requires 2 states to compute the dynamics, so overall we need $2n$ state samples for bounding $f([a]_\varepsilon)$. Also note that to cover a hyperbox we need to consider the gradient direction for the most rapid change in it.

When we learn barriers on partial state dimensions, the range in unobserved dimensions implicitly affects the bounds on $f(a)$. For a full state x, we partition it and write the observation as a and unobserved dimensions as y. To get the dynamics bounds, besides $[a]_\varepsilon$, we also need to consider a hypterbox around y, $[y]_\beta$. To do so, when finding dynamics in the observation space, after using kNN to find the full state, we sample $\varepsilon e_{a,i}$ and $\beta e_{y,i}$ for each coordinate, to find the dynamics bounds $f([a]_\varepsilon, [y]_\beta)$. At deployment, we need to verify that a new state x''s unobserved dimensions y' must be within the corresponding $[y]_\beta$ that is used during certification.

5.3 Retraining Neural Barrier Function

To further improve our neural barrier function, we can retrain it by adding the points that violate the third Barrier condition: $\mathsf{L}_f B(x) > 0$, $\forall x$ s.t. $B(x) = 0$. One idea is to leverage the verification bounds in the earlier chapter to first find the violated observations: for each observation x, it is a violation if (1) $B_L(x) < 0$ and $B_L(x) > 0$, and (2) $\mathsf{L}_f B(x) > 0$.

A violation can be found with the following procedues. First, use methods described in Section 5.1 to find the interval bounds of barrier function values B_U and B_L for states in the state space, and those with $B_U > 0$ and $B_L < 0$ are identified with boundary states x_b. We then calculate for the lie derivatives of the boundary states, which consists of the bounds of barrier function network derivatives and the bounds of dynamics. The former can be found using the methods in Section 5.1. To find dynamics bounds, we firstly need to generate the dynamics of

the states, which is further described in Section 6.3. Then, using the method from Section 5.2, we can get the bounds on the dynamics. With these information, we are able to compute the upper and lower bounds of $\mathsf{L}_f B(x_b)$ in an ε box for all boundary states, and identify the ones with upper bounds $\left[\mathsf{L}_f B(x_b)\right]_U > 0$ as violations. For each violation state, it is labeled using a KNN approach. Then, define $\rho \in (0.5, 1]$ for the threshold of KNN classification. For each observation, find k nearest labels from $X_s \cup X_u$. If more than ρk neighbors are from X_s or X_u, it is added into X_s or X_u, respectively. Otherwise the violation set remains the same. We can then retrain the neural barrier functions with the updated safe and unsafe sets by minimizing the loss function in Equation 4.1. We continue doing so until the number of violations is smaller than some threshold.

Chapter 6

Experiments

6.1 Environments

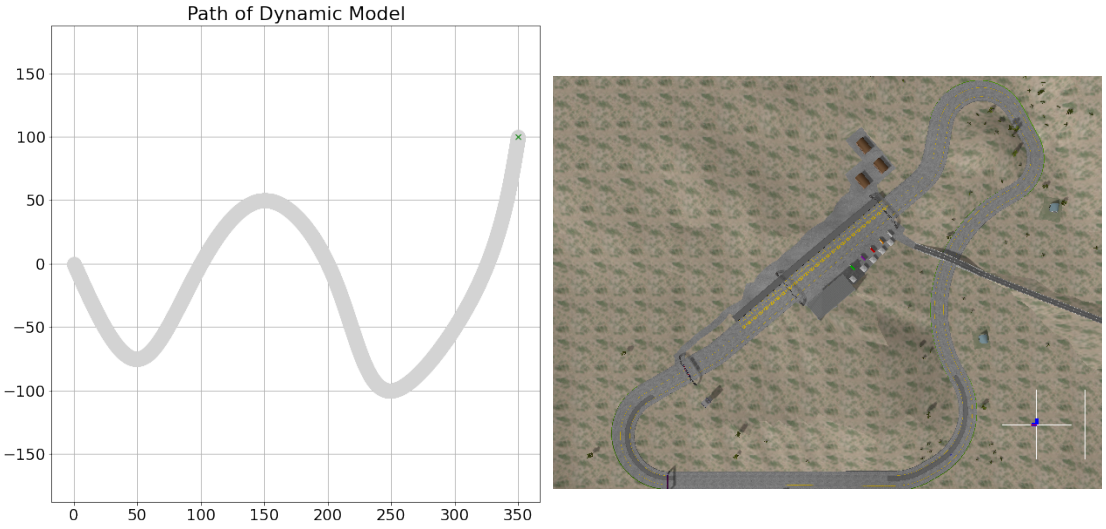


Figure 6.1: Evaluation environments. Left:A path for the kinematic and dynamic model simulation environments. Right: Path in the TORCS vehicle dynamics simulation environment.

We tested the training and certification pipeline on three environments: (a) 2-D path-tracking kinematic bicycle model with no-slip assumption (abbreviated as kinematic model below), (b) 2-D path-tracking dynamic bicycle model with tire-slip lateral control (abbreviated as dynamic model below), and (c) TORCS racing environment, which is the most realistic

environment among the three compared to the real world. Figure 6.1 shows the paths of dynamic model and TORCS. The path used for kinematic model is very similar, but with x and y in smaller ranges (i.e. shorter tracks with the same shape). In (a), the kinematic model contains three state dimensions: cross-track error, angle error and longitudinal speed. In (b), the dynamic model contains two additional state dimensions: lateral speed and vehicle yaw (spin) rate. In (c), the state information for TORCS is more complex and can be found in [48]. The kinematic model assumes the tires do not have lateral slip, while the dynamic model and TORCS environment take inertial effect and lateral slip into account. In addition, in kinematic and dynamic model, we have the ability to arbitrarily sample the vehicle states (arbitrary reset), while in TORCS we are limited to sample a small range of initial configurations which poses additional difficulty for the sampling procedures. The action space in the three environments is the same.

6.2 Training the Neural Controller

We use a two-layer fully connected neural network, with 128 hidden nodes in each layer and ReLU activations, to represent the control policy. The neural controllers for kinematic and dynamic models use full state information, while the one for TORCS used selected state including distance error, angle error and longitudinal speed. The distance error measures how far the vehicle deviates from the track, and the angle error measures the difference between the track's direction and the vehicle's heading direction. We used standard policy optimization methods (Proximal Policy Optimization [46] and Soft Actor-Critic [7]) to train the neural controllers. We used the following reward function to train the neural controller with PPO and SAC:

$$r_t = \begin{cases} -|e_t^s| - |e_t^p| - |e_t^a|, & s_t \text{ is not terminal} \\ 100, & s_t \text{ is goal} \\ -100, & s_t \text{ is out of the track} \end{cases} \tag{6.1}$$

Where e_t^s, e_t^p, e_t^a represent speed error, distance error and cross-angle error, respectively.

6.3 Evaluating Lie Derivatives in Different Environments

As mentioned above, certifying the control barrier functions requires computing the Lie derivatives of the states on the boundary, which needs the information of the dynamics of these points. This could be challenging when we only have partial state information and when the environment does not allow resetting to arbitrary states. To overcome the challenges, we propose the following three methods that enable us to estimate the dynamics of the grid.

Full-State Barriers with Arbitrary Reset When the barrier function operates on full state information, finding the dynamics is straightforward: we can set the environment to the desired states and simulate one time step forward with the action from neural controller to obtain the dynamics. This is the method used to find the dynamics of the kinematic model.

Partial-State Barriers with Arbitrary Reset We observed that in the dynamic model, when a sharp corner is encountered with high speed, it is necessary for the vehicle to decelerate in order not to deviate from the track and crash. Plotting the trajectories in the distance error, angle error and (longitudinal and lateral) speeds, we get the Figure 6.2. In practice, it is hard to train a neural network to output the shape like the one shown in the figure. In addition, since we need to verify for each cell in the grid, the computation complexity increases exponentially as more dimensions are added. This makes it infeasible to train and verify a barrier function on the full state for the dynamic model. Thus, it is important to design methods for capturing the barrier in low-dimensional projections of the states. Intuitively, it is feasible because even though the vehicle has high-dimensional states, its key safety properties are typically observed in the 2D or 3D physical environment (not hitting obstacles on a 2D plane) and the certificate can be low-dimensional.

The challenge then comes in how to find the dynamics with missing state information.

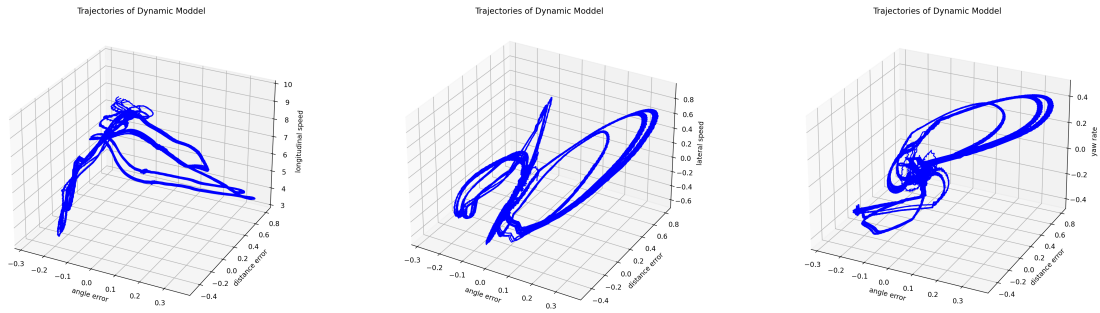


Figure 6.2: Trajectories of neural controller on dynamic model. Left: angle error, distance error and longitudinal speed. Right: angle error, distance error and lateral speed

By further examining longitudinal speed, lateral speed and yaw rate in Figure 6.2, it can be shown that at the boundaries where distance error and angle error are large, the other dimensions are within a small range.

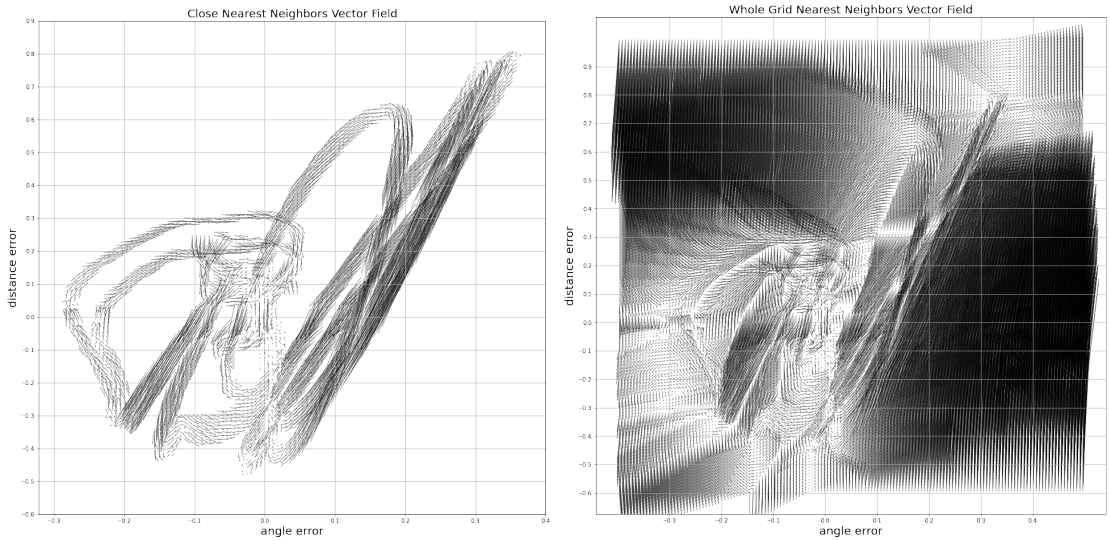


Figure 6.3: Vector field of the system dynamics calculated with nearest neighbor approach for the dynamic model.

Moreover, in Figure 6.3 we plot the vector field of the the dynamic model. The left figure shows the dynamics of the points that are close to the collected trajectories, and the right figure

shows the dynamics of the whole grid (note that we do not have to find the dynamics for the whole grid: here we are just computing it to show the calculated dynamics). As can be shown, at the boundary, the arrows tend to converge to the center, implying the barrier behavior when we only consider distance and angle errors.

Based on the observations above, we proposed a method of using Nearest Neighbor to find the dynamics in unobserved regions.

Suppose the full states of the environment is $x \in \mathbb{R}^n$ and we want to train a barrier function with partial observation $o \in \mathbb{R}^m$, where $m < n$. After collecting many trajectories with full state in $\mathbb{R}^n$, we select the observation dimensions we want to train barrier function on and form an observation set O_t in $\mathbb{R}^m$. Then, we can find the system dynamics for a boundary observation in the observation space $\mathbb{R}^m$ by finding its nearest neighbor in O_t. We then find the nearest neighbor's corresponding full state in $\mathbb{R}^n$, and replace the observation dimensions with the one from the boundary observation. Using the newly generated full state, we simulate one step forward to find the system dynamics of the boundary state. This is the technique we use to find the dynamics for the dynamic model.

Partial-State Barriers without Arbitrary Reset Without the ability to set the system to arbitrary state, we rely solely on the nearest neighbor approach to find the system dynamics for the boundary observations. Here, for each point in the grid, we find its nearest neighbor in the collected trajectories, and use that neighbor's dynamics as the dynamics for the boundary observation. Although the method leads to some inaccuracy for observations that are far from the observed samples, since we only need accurate dynamics around boundary and the certification step utilizes a range of dynamics, the verified barrier function using this method is still valid. We use this method to find the dynamics of the TORCS racing environment

6.4 Training the Barrier Function

To train the control barrier function, we firstly obtain a neural controller that performs well in the tracking environment, collect safe samples and generate unsafe samples. We then begin the training, verification and retraining loop of barrier function.

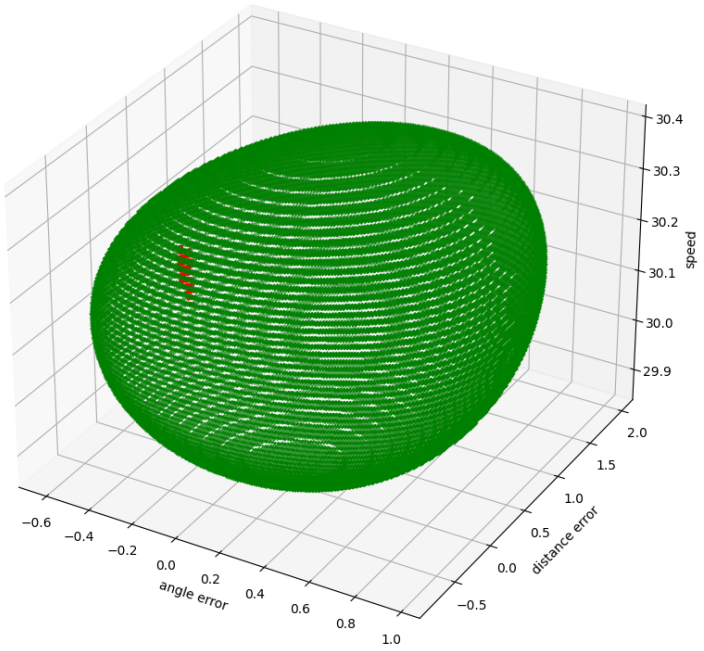
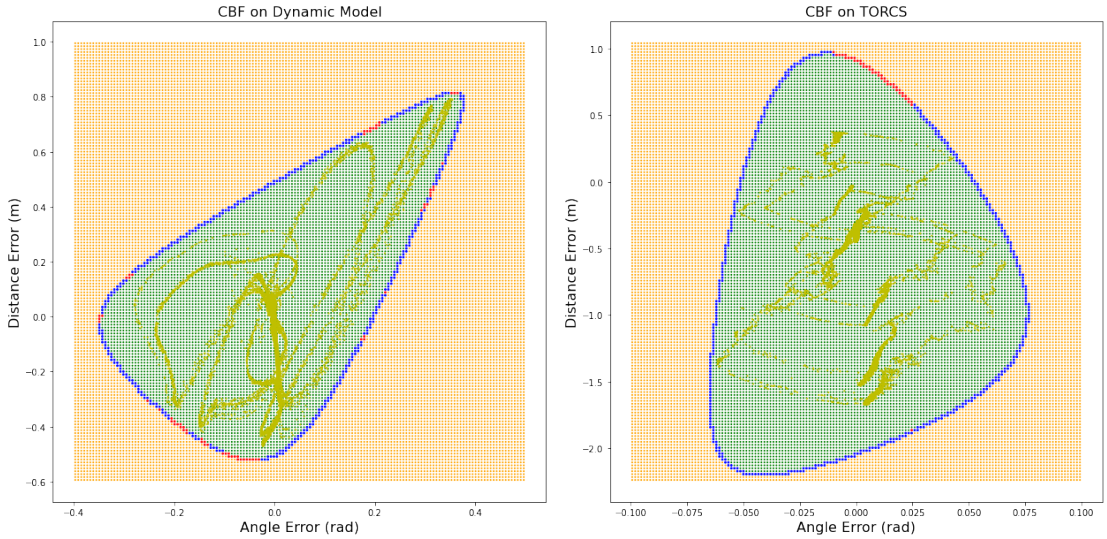


Figure 6.4: Full-state control barrier function of kinematic model with target speed of $30m/s$. The points show barrier function boundary, and the red points represent the region with lie derivative violations. The violation rate on the boundary is 0.95%

Certification Results on the Kinematic Model. In the kinematic model, we have a simple bicycle vehicle model without wheel slipping. In this setting we can arbitrarily reset the vehicle to any state, allowing us to approximate the dynamics and Lie derivatives at any state.

Figure 6.4 shows the full-state barrier function on the Kinematic Model. In this experiment, we let the vehicle follow a speed of $30m/s$. The points are boundary states of barrier function where $B_L(x) < 0$ and $B_U(x) > 0$. The green points are where the boundary conditions are satisfied (i.e. $\left[\mathsf{L}_f B(x)\right]_U < 0$ within the range of the ε box, and the red points show violations. In this

barrier function result, the violation rate among the boundary is 0.95%, which sufficiently certified that a vehicle starting from inside the barrier function is guaranteed to never leave the safety set except for the violation region.

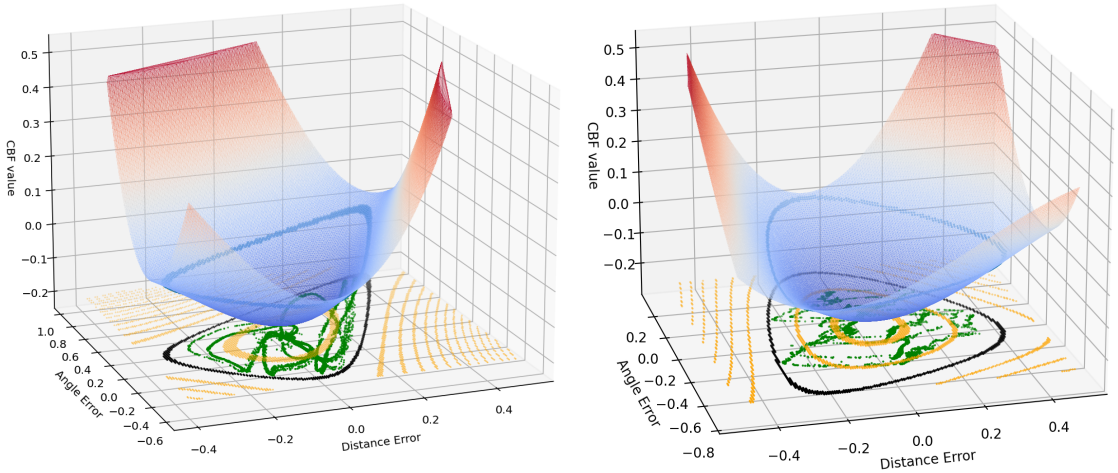


(a) Target: $10m/s$. Boundary violation: 13.6% (b) Target: $20m/s$. Boundary violation: 8.86%

Figure 6.5: Barrier function of dynamic model and TORCS environment, certified on partial observation with distance and angle errors. The blue points are the barrier, and yellow points are the collected (safe) observations.

Partial-State Barriers with Arbitrary Reset To understand safety of neural controllers in more realistic setup, we trained and verified the barrier function on dynamic model, which contains wheel slipping so that the vehicle would crash out of the track if a hard turn was applied under high speed. Because of the deceleration at sharp corners and increased dimensionalities, it is infeasible to train and verify a barrier function on the full state space. Thus, we trained and verified barrier function with partial state observation in the 2D space of distance error and angular error.

In the dynamic model, we let the vehicle follow a target speed of $10m/s$. Figure 6.5a and Figure 6.6a show the certification results. The final barrier function separates the safe trajectories from the unsafe observations, and on the boundary, and 86.4% boundary states satisfy the lie



(a) (b)

Figure 6.6: 2D barriers of dynamic model and TORCS environment. The surface plot shows the value of learned barrier function, and the x-y plane shows the level sets, with vehicle trajectories within the safe set. The zero-level set is colored in black.

derivative requirements. The decrease of the number of certified boundary points is due to two factors. First, as can be seen in Figure 6.1, the curvatures of corners are very large, making it more delicate for the barrier function to satisfy boundary conditions. Second, although we have the ability of setting the vehicle to arbitrary state, since we are training on partial observation, the additional dimensions has to be estimated using nearest neighbor, making the dynamics error range larger (as opposed to the full state barrier function on kinematic model, where there is no such information that requires estimation).

Certification Results on the Dynamic Model with Lateral Slip. In a more realistic setting, we train and verify the barrier function on dynamic model, which contains wheel slipping - the vehicle can crash if a hard turn was applied under high speed in this model. We train barrier functions with partial state observation in the 2D space of distance and angular errors. We can sample arbitrary states in this simulation for estimating the system dynamics.

In the dynamic model, we let the vehicle follow a target speed of $10m/s$. In Figure 6.5a,

we show the trained barrier projected in 2D, and in Figure 6.5a, we show the value of the barrier function, with level sets plotted in the x-y plane. The final barrier separates the safe trajectories from the unsafe observations, and 86.4% boundary states satisfy the lie derivative requirements. The decrease of the number of certified boundary points is because we are training on partial observation, and the range in unobserved dimensions increases dynamics bounds, making it harder to satisfy Lie derivative requirements on the boundary.

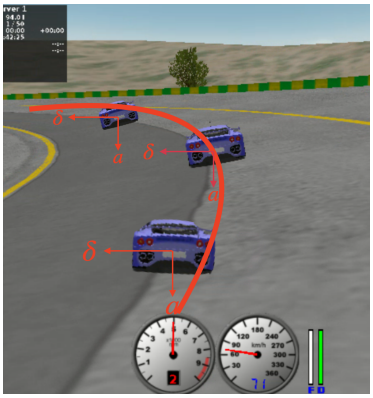
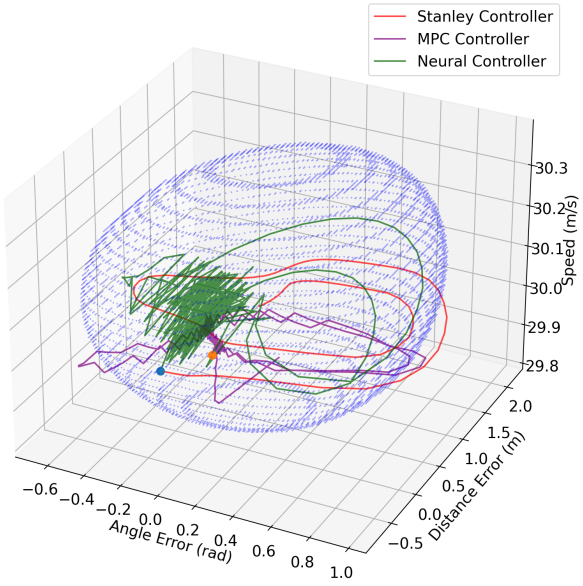


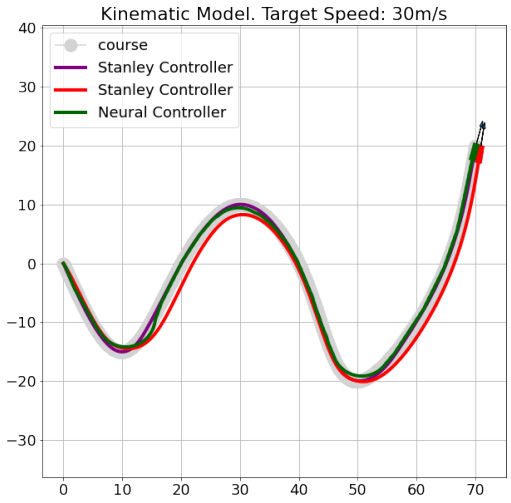
Figure 6.7: Illustration of the barrier on TORCS model: when the vehicle is approaching the boundary, the neural controller applies brake and steers the vehicle to the side where the barrier function value decreases.

Certification Results in the TORCS Simulator. The third environment we consider is TORCS racing environment, where the car would slip under high speeds, and we cannot reset the vehicle to arbitrary states, making it hard to precisely estimate system dynamics on all states. We train and certify a neural barrier function on TORCS using partial observation with angle and distance errors. In this experiment, we let the vehicle to follow a speed of $20m/s$. Figure 6.5b and Figure 6.6b show the full-state barrier function for the TORCS environment, with a violation rate of 8.86%. The lack of the ability to reset to arbitrary states makes it only possible to use nearest neighbor to estimate the dynamics, decreasing boundary satisfiability. However, the controller still demonstrates great barrier behaviors. Figure 6.7 shows one instance where the vehicle is heading toward the barrier, with the vehicle in the middle on the barrier. In these short clips,

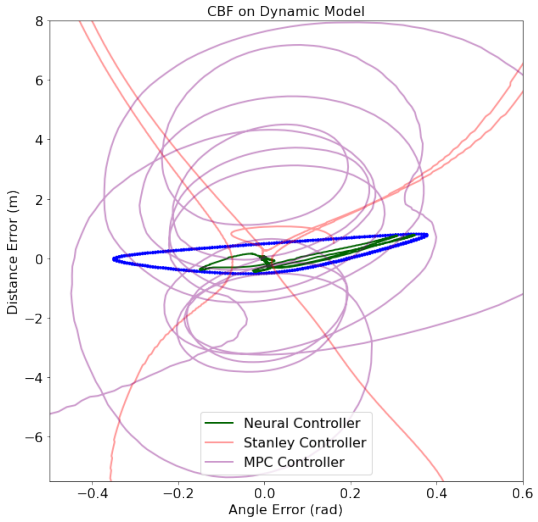
the controller is continuously applying hard brake while turning the wheel on the left, to keep it inside the learned barrier.



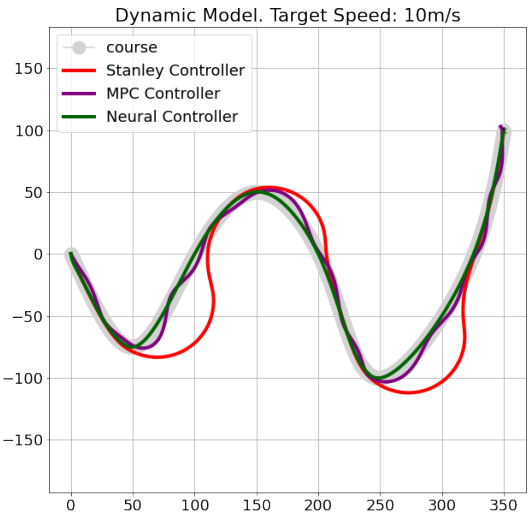
(a) Stanley, MPC and neural controller in kinematic model, with respect to the learned neural barrier.



(b) Tracking performance of the three controllers in the kinematic model environment.



(c) 2D projections of the vehicle trajectories under the three controllers in the dynamic model.



(d) Tracking performance of the three controllers in the dynamic model environment.

Figure 6.8: Performance comparison among Stanley controller, MPC and neural controller in path tracking tasks.

6.5 Consistency of Performance with Learned Barriers

In this section, we further validate that the learned neural barriers certify safety of the neural controllers. In Figure 6.8, we see that the vehicle trajectories using the neural controller are indeed contained in the safe set defined by the neural barrier. In contrast, we show trajectories using the Stanley and MPC controllers violate the safety barriers. Figure 6.8b and 6.8d show the control behavior of the neural, Stanley and MPC controllers on the evaluation path, for kinematic and dynamic models. Figure 6.8a and 6.8c compare the trajectories on the dimensions where the barrier functions are defined. For both models, the trajectory from the neural controller never leaves the barrier, while the vehicle trajectories under the other controllers do not respect the learned neural barriers and escape the corresponding safe set.

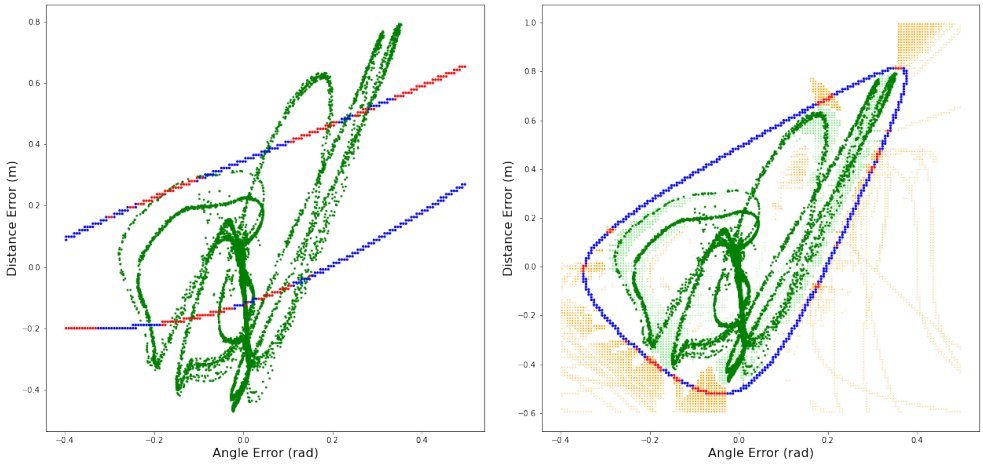


Figure 6.9: Illustration of continuous training. Left: initial barrier (blue and red) and original safe data (green); Right: final barrier after continuous training and relabeled violations (safe: light green; unsafe: yellow).

6.6 Importance of retraining

In Figure 6.9, we show how the barrier for dynamic model changes during continuous training, where we train until convergence, relabel violations as safe or unsafe data, and retrain

the barrier function with amended dataset. As shown from the left figure, the barrier function does not capture the true shape of the barrier, and there are a large portion (more than 50%) of violations on the learned barrier. For final barrier function as shown on the right, we can see the added safe data are close to the collected trajectory, while the added unsafe data are far from it, which also gives a barrier function with desired shape and reduces violation rate.

Chapter 7

Conclusion

We proposed new methods for training and certifying neural barrier function for neural controllers in the path-tracking control problem for self-driving vehicles. We described methods for generating safe and unsafe data and the iterative training of barrier function, as well as certification methods that analyze the numerical robustness of the neural barrier functions for checking that the lie derivative requirements are satisfied around sampled states. By conducting experiments on three environments ranging from simple to realistic dynamics, we showed the feasiblilty of the proposed approach for quantifying the safety of neural controllers. In future work we believe the neural controllers can be tested on physical vehicles with the neural barriers serving as runtime monitoring of safe control of the vehicles. We believe the learned neural barriers provide an important interface between low-level control with planning and perception layers for autonomous driving, and can provide useful information for safe human-robot interaction in these systems.

The thesis is currently being prepared for submission for publication of the material. It is a joint work with Tsui-Wei (Lily) Weng and Sicun Gao. The thesis author was the primary author of this material.

Bibliography

[1] J. M. Snider, "Automatic steering methods for autonomous automobile path tracking," Tech. Rep. CMU-RI-TR-09-08, Carnegie Mellon University, Pittsburgh, PA, February 2009.

[2] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[3] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *2015 IEEE Intelligent Vehicles Symposium, IV 2015, Seoul, South Korea, June 28 - July 1, 2015*, pp. 1094–1099, 2015.

[4] W. Schwarting, J. Alonso-Mora, L. Pauli, S. Karaman, and D. Rus, "Parallel autonomy in automated vehicles: Safe motion generation with minimal intervention," in *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pp. 1928–1935, 2017.

[5] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. M. Oakley, M. Palatucci, V. R. Pratt, P. Stang, S. Strohband, C. Dupont, L. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. R. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. V. Nefian, and P. Mahoney, "Stanley: The robot that won the DARPA grand challenge," *J. Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms.," *CoRR*, vol. abs/1707.06347, 2017.

[7] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 1856–1865, 2018.

[8] T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, D. Boning, I. S. Dhillon, and L. Daniel, "Towards fast computation of certified robustness for relu networks," *ICML*, 2018.

[9] L. Weng, P.-Y. Chen, L. Nguyen, M. Squillante, A. Boopathy, I. Oseledets, and L. Daniel, "Proven: Verifying robustness of neural networks with a probabilistic approach," *ICML*, 2019.

[10] A. Girard and G. J. Pappas, "Verification using simulation," in *Hybrid Systems: Computation and Control* (J. P. Hespanha and A. Tiwari, eds.), (Berlin, Heidelberg), pp. 272–286, Springer Berlin Heidelberg, 2006.

[11] A. Puri, P. Varaiya, and V. Borkar, "epsilon;-approximation of differential inclusions," 1995.

[12] E. Asarin, T. Dang, and A. Girard, "Reachability analysis of nonlinear systems using conservative approximation," in *Hybrid Systems: Computation and Control* (O. Maler and A. Pnueli, eds.), (Berlin, Heidelberg), pp. 20–35, Springer Berlin Heidelberg, 2003.

[13] Zhi Han and B. H. Krogh, "Reachability analysis of nonlinear systems using trajectory piecewise linearized models," in *2006 American Control Conference*, pp. 6 pp.–, 2006.

[14] M. Althoff, O. Stursberg, and M. Buss, "Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization," in *2008 47th IEEE Conference on Decision and Control*, pp. 4042–4048, 2008.

[15] T. Dang, O. Maler, and R. Testylier, "Accurate hybridization of nonlinear systems," in *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2010, Stockholm, Sweden, April 12-15, 2010* (K. H. Johansson and W. Yi, eds.), pp. 11–20, ACM, 2010.

[16] A. Chutinan and B. H. Krogh, "Computational techniques for hybrid system verification," *IEEE Transactions on Automatic Control*, vol. 48, no. 1, pp. 64–75, 2003.

[17] C. J. Tomlin, I. Mitchell, A. M. Bayen, and M. Oishi, "Computational techniques for the verification of hybrid systems," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 986–1001, 2003.

[18] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin, "Fastrack: A modular framework for fast and guaranteed safe motion planning," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 1517–1522, 2017.

[19] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *2007 American Control Conference*, pp. 2296–2301, 2007.

[20] V. K, M. Ambalal Sheta, and V. Gumtapure, "A comparative study of stanley, lqr and mpc controllers for path tracking application (adas/ad)," in *2019 IEEE International Conference on Intelligent Systems and Green Technology (ICISGT)*, pp. 67–674, 2019.

[21] S. Dominguez, A. Ali, G. Garcia, and P. Martinet, "Comparison of lateral controllers for autonomous vehicle: Experimental results," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1418–1423, 2016.

[22] S. Prajna, A. Jadbabaie, and G. J. Pappas, "A framework for worst-case and stochastic safety verification using barrier certificates," *IEEE Trans. Autom. Control*, vol. 52, no. 8, pp. 1415–1428, 2007.

[23] P. Wieland and F. Allgöwer, "Constructive safety using control barrier functions," in *Proc. IFAC Symp. Nonlin. Control Syst.*, (Pretoria, South Africa), pp. 462–467, August 2007.

[24] A. D. Ames, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs with application to adaptive cruise control," in *Proc. Conf. Decis. Control*, (Los Angeles, CA,), pp. 6271–6278, December 2014.

[25] P. A. Parrilo, *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. PhD thesis, California Institute of Technology, 2000.

[26] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Trans. Autom. Control*, vol. 62, no. 8, pp. 3861–3876, 2017.

[27] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *2019 18th European Control Conference (ECC)*, (Naples, Italy), pp. 3420–3431, June 2019.

[28] X. Xu, J. W. Grizzle, P. Tabuada, and A. D. Ames, "Correctness guarantees for the composition of lane keeping and adaptive cruise control," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 3, pp. 1216–1229, 2017.

[29] L. Wang, D. Han, and M. Egerstedt, "Permissive barrier certificates for safe stabilization using sum-of-squares," in *2018 Annual American Control Conference (ACC)*, pp. 585–590, IEEE, 2018.

[30] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, pp. 3387–3395, 2019.

[31] R. Cheng, A. Verma, G. Orosz, S. Chaudhuri, Y. Yue, and J. Burdick, "Control regularization for reduced variance reinforcement learning," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 1141–1150, PMLR, 2019.

[32] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *ICCAV*, 2017.

[33] M. Hein and M. Andriushchenko, "Formal guarantees on the robustness of a classifier against adversarial manipulation," in *NIPS*, 2017.

[34] T.-W. Weng, H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, and L. Daniel, "Evaluating the robustness of neural networks: An extreme value theory approach," *ICLR*, 2018.

[35] J. Z. Kolter and E. Wong, "Provable defenses against adversarial examples via the convex outer adversarial polytope," *ICML*, 2018.

[36] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," in *NeurIPS*, 2018.

[37] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," in *NeurIPS*, 2018.

[38] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," in *NeurIPS*, 2018.

[39] A. Raghunathan, J. Steinhardt, and P. S. Liang, "Semidefinite relaxations for certifying robustness to adversarial examples," in *NeurIPS*, pp. 10877–10887, 2018.

[40] A. Boopathy, T.-W. Weng, P.-Y. Chen, S. Liu, and L. Daniel, "Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks," in *AAAI*, 2019.

[41] C.-Y. Ko, Z. Lyu, T.-W. Weng, L. Daniel, N. Wong, and D. Lin, "Popqorn: Quantifying robustness of recurrent neural networks," *ICML*, 2019.

[42] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[43] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *ICML 2016*, pp. 1928–1937, 2016.

[44] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization," in *ICML 2015*, pp. 1889–1897, 2015.

[45] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Second-order optimization for deep reinforcement learning using kronecker-factored approximation," in *NIPS 2017*, pp. 5285–5294, 2017.

[46] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[47] W. Jin, Z. Wang, Z. Yang, and S. Mou, "Neural Certificates for Safe Control Policies," *arXiv e-prints*, p. arXiv:2006.08465, June 2020.

[48] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Simulated car racing championship: Competition software manual," *CoRR*, vol. abs/1304.1672, 2013.