Neural Control Barrier Functions
from
Physics Informed Neural Networks

A Thesis submitted for the completion of
requirements for the degree of

Bachelor of Science
(Research)

by

Shreenabh Moujesh Agrawal
Undergraduate Programme
Indian Institute of Science



Under the supervision of

Dr. Shishir N. Y. Kolathaya
Robert Bosch Centre for Cyber Physical Systems (RBCCPS),
Indian Institute of Science

Dr. G. R. Jayanth
Department of Instrumentation and Applied Physics (IAP),
Indian Institute of Science

Certificate

This is to certify that the Bachelor's thesis titled *"Neural Control Barrier Functions from Physics Informed Neural Networks"* submitted by **Shreenabh Moujesh Agrawal** (**Sr No. 11-01-00-10-91-21-1-20412**) to the Indian Institute of Science, Bangalore, towards partial fulfilment of requirements for the award of the degree of **Bachelor of Science (Research) in Physics**, is a record of bona fide work carried out by him under my supervision and guidance during the Academic Year, 2024–25.

Advisor (Physical Sciences)

Dr. G. R. Jayanth
Associate Professor
Department of Instrumentation and Applied Physics (IAP)
Indian Institute of Science (IISc), Bangalore

Signature: G.R Jay

Date: 11/04/25

Place: IISc, Bangalore

Co-Advisor

Dr. Shishir N. Y. Kolathaya
Assistant Professor
Robert Bosch Centre for Cyber Physical Systems (RBCCPS)
Indian Institute of Science (IISc), Bangalore

Signature: Shishir N Y
Shishir Nadubettu Yadukumar Kolathaya (Apr 12, 2025 17:41 GMT+5.5)

Date: _____

Place: IISc, Bangalore

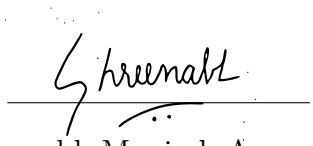
Declaration

I certify that:

1. The work contained in this report has been done by me under the guidance of my supervisor(s).
2. The work has not been submitted to any other Institute for any degree or diploma.
3. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
4. Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources or people, I have given due credit to them by citing them in the text of the thesis and listing their details in the references.

Date: 12 April 2025

Place: IISc, Bangalore

Signature: 

Name: Shreenabh Moujesh Agrawal

SR No.: 11-01-00-10-91-21-1-20412

Acknowledgements

I would like to sincerely thank Prof. Shishir Nadubettu Yadukumar Kolathaya for his continuous mentorship and guidance throughout the duration of this project. His insightful discussions, continued encouragement, and generous provision of laboratory resources have greatly shaped my understanding and approach to this research.

I am very grateful to Prof. G. R. Jayanth, who provided the physics perspective on this problem and handled the administrative logistics of the thesis. My sincere gratitude extends to Prof. Pushpak Jagtap, whose mentorship in previous projects at RBCCPS cultivated my foundational knowledge in robotics safety, significantly enabling me to undertake and complete this thesis.

I am also grateful to Manan Tayal and Aditya Singh from the Stochastic Robots Lab for their discussions, helping me structure the problem-solving approach, and their hands-on assistance during hardware experiments in the lab. Furthermore, I would like to acknowledge all the staff at the Stochastic Robots Lab, the Physics Department, the Undergraduate Office, and the numerous professors and mentors at IISc whose direct and indirect efforts have led to this thesis submission.

Finally, my heartfelt thanks goes out to my family and friends for their constant support and encouragement. A special thanks to my dad for all the pizzas, paninos, and popsicles growing up.

Abstract

As autonomous systems become increasingly prevalent in daily life, ensuring their safety is paramount. Recent work has shown that Control Barrier Functions (CBFs) provide verifiable guarantees of safety and have consequently emerged as effective tools for this task. Nevertheless, manually designing them for specific applications remains a significant challenge. With the advent of deep learning techniques, recent research has explored synthesizing CBFs using neural networks—commonly referred to as neural CBFs. This paper introduces a novel class of neural CBFs leveraging a physics-inspired neural network framework by incorporating Zubov's Partial Differential Equation (PDE) within the context of safety. The proposed approach is scalable and suitable for synthesizing neural CBFs even for systems with high-dimensional state spaces, as demonstrated by an aerial navigation example. By utilizing neural reciprocal CBFs as described in this work, users can modify the "conservativeness" of the safe regions by appropriately selecting corresponding sublevel sets. To validate the effectiveness of the approach, case studies are presented on three different systems: an inverted pendulum, autonomous ground navigation, and aerial navigation in obstacle-laden environments.

Note 1: A research paper that captures the key elements of this thesis has been submitted to *2025 64th IEEE Conference on Decision and Control (CDC)* [Manuscript 1184, Submitted April 3, 2025].

Note 2: The preliminaries, related work discussion, and methodological analysis are presented in comprehensive detail with appropriate citations. The most critical sections of Chapters 2, 3, and 4 are highlighted in bold in the table of contents. These sections maintain internal coherence and are self-contained, allowing the reader to sequentially understand the primary contributions of the thesis.

Contents

Certificate i

Declaration ii

Acknowledgements iii

Abstract iv

1 Introduction 1

2 Preliminaries 3

2.1 Control Lyapunov Functions 3

2.2 Barrier Functions 4

2.2.1 Reciprocal BF 4

2.2.2 Zeroing BF 5

2.2.3 Relationships between RBF and ZBF 5

2.3 Control Barrier Functions 6

2.3.1 **Reciprocal CBFs** 6

2.3.2 Zeroing CBFs and Higher Relative Degree 8

2.4 Optimization-based Control using QP 9

2.4.1 CLF QP 9

2.4.2 **CBF QP** 10

2.4.3 CLF-CBF QP 11

3 Problem Formulation 12

3.1 Lyapunov's Characterization for Domain of Attraction 12

3.2 Zubov's Characterization for Domains of Attraction 13
3.3 Equivalence of Characterizations & Benefits 14
3.4 **Zubov's Characterization for Safe Regions** 18
3.5 **Construction of W_N** 19

4 Methodology **21**
4.1 **Neural Network Training** 21
4.2 **Designing a Controller** 22
4.3 Error Estimates and Convergence Analysis 24

5 Experimental Results **27**
5.1 Inverted Pendulum 27
5.2 Autonomous Ground Navigation 29
5.2.1 Simulation Results 29
5.2.2 Hardware Results 31
5.3 Aerial navigation in obstacle-rich environments 32
5.3.1 Reference Controller Design 34
5.3.2 Barrier-Based Controller Integration 36
5.3.3 Simulation Setup and Results 36

6 Conclusion and Future Work **40**
6.1 Conclusions 40
6.2 Future work 40

A Code Implementations **42**
A.1 Algorithm 1: Unicycle Implementation 42
A.2 Algorithm 2: Unicycle Implementation 46

Chapter 1

Introduction

Ensuring safety is a primary challenge in control systems, especially in autonomous applications where real-time guarantees are essential. Any safety violation can have severe consequences; therefore, designing control strategies that both safeguard operations and maintain efficiency is imperative.

To address this challenge, several paradigms have been developed. Constrained Reinforcement Learning (CRL) methods [1, 2] embed safety constraints within a reinforcement learning framework to learn safe policies in a data-driven manner. However, due to their reliance on exploration, these methods may not consistently enforce strict safety guarantees. Alternatively, Hamilton-Jacobi (HJ) reachability analysis [3, 4, 5] provides rigorous worst-case assurances by computing backward-reachable sets that delineate a safe operating domain. Despite its theoretical robustness, the substantial computational burden of HJ reachability makes it less practical for real-time, high-dimensional systems.

A particularly promising approach involves the use of Control Barrier Functions (CBFs) [6], which offer an effective means of synthesizing safe control laws for control affine systems [7]. By formulating the control design as Quadratic Programs (QPs), which can be solved at high frequencies using modern optimization solvers, CBFs have been successfully applied to various safety-critical tasks such as adaptive cruise control [6], aerial maneuvers [8, 9], and legged locomotion [10, 11]. Nonetheless, the safety and performance of these applications are highly dependent on the chosen CBF, and traditional synthesis methods—such as sum-of-squares techniques [12, 13]—often struggle with high-dimensional state spaces.

In recent years, the universal approximation capabilities of neural networks have spurred interest in neural network-based barrier function (NCBF) synthesis. A diverse range of approaches has emerged, including methods that leverage expert demonstrations [14], SMT-based techniques [15, 16], mixed-integer programming [17], and nonlinear programming [18]. Recent work has also introduced specialized loss functions for training

NCBFs [19, 20, 21, 22]. However, while methods that produce formally verified CBFs [23, 24] offer strong guarantees, they often lack scalability when applied to high-dimensional systems. Moreover, none of the previously discussed approaches provide the flexibility to select a user-defined sub-level set, which is essential for tailoring the safe region to particular requirements.

Building on recent advances in Neural Control Lyapunov Function (NCLF) synthesis using Physics-Informed Neural Networks (PINNs) with Zubov's method [25, 26, 27, 28], this thesis systematically investigates a novel framework for learning Neural Control Barrier Functions (NCBFs) through a Zubov PDE-based construction. Incorporating Zubov's PDE during training leads to the development of maximal neural Lyapunov functions with verifiable domains of attraction that closely approximate analytical solutions [26, 27].

Prior studies in stability analysis have demonstrated that Zubov's theorem provides a useful characterization of domains of attraction by transforming the Lyapunov function learning problem into a bounded PDE problem. Constructing a transformed Lyapunov function that is guaranteed to be bounded overcomes numerical issues encountered during synthesis, improves training stability, and allows for user-adjustable sub-level sets according to specific requirements. Recognizing that similar challenges exist in synthesizing Reciprocal Control Barrier Functions (RCBFs), this work introduces a PINN-based methodology to learn them, thereby integrating the advantages of Zubov's approach into the training process.

The primary contributions of this thesis are as follows:

- A reformulation of Zubov's PDE in the context of safety to synthesize neural CBFs using PINNs and design a safe controller through its QP formulation.
- The proposed framework allows for flexible sub-level set selection, thus allowing users to tailor the safe region to specific requirements.
- The developed algorithms demonstrate efficient training, exhibiting rapid convergence and high sample efficiency with only 10,000 random samples and 2,000 training epochs across diverse environments.
- Validation of the approach through experiments on an inverted pendulum, autonomous ground navigation, and aerial navigation in obstacle-rich environments, demonstrating scalability to high-dimensional systems.

Chapter 2

Preliminaries

2.1 Control Lyapunov Functions

In this section, we introduce *Control Lyapunov Functions* (CLFs) for the deterministic control system described in (2.6). A continuous function $V : D \to \mathbb{R}_{\geq 0}$ is typically called a Lyapunov function if it is positive definite and decreases along system trajectories; one therefore obtains *stability* by ensuring $\dot{V}(x)$ is suitably negative. In the *control* setting, we seek an input u that enforces negativity of $\dot{V}(x)$.

Specifically, define:

$$\dot{V}(x,u) \;=\; \nabla V(x)^{\top}\Big(f(x) + g(x)\,u\Big), \tag{2.1}$$

where $\nabla V(x) \in \mathbb{R}^n$. A continuous function V is called a *control Lyapunov function (CLF)* if there exists a class $\mathcal{K}$ function $\gamma(\cdot)$ such that

$$\inf_{u \in \mathbb{U}} \dot{V}(x,u) \;\leq\; -\gamma\Big(V(x)\Big) \quad \text{for all } x \in D. \tag{2.2}$$

Here, γ maps $\mathbb{R}_{\geq 0}$ to $\mathbb{R}_{\geq 0}$, is strictly increasing, and $\gamma(0) = 0$. The *feasibility* of (2.2) ensures that there is a feedback control law that renders $\dot{V}(x)$ negative whenever $x \neq 0$, thereby stabilizing the system to the set $\{x : V(x) = 0\}$ (often the origin).

Definition 2.1.1 (Admissible Stabilizing Inputs). Given a CLF V, define the set of all *stabilizing* control inputs

$$K_{\text{clf}}(x) \;=\; \Big\{u \in \mathbb{U} \;\Big|\; \dot{V}(x,u) \;\leq\; -\gamma\Big(V(x)\Big)\Big\}. \tag{2.3}$$

Any (Lipschitz continuous) $u(x) \in K_{\text{clf}}(x)$ for all x guarantees that $V(x(t))$ converges to zero as $t \to \infty$, thereby stabilizing the closed-loop system [29, 27].

A special case arises with an *exponentially stabilizing* CLF (ES-CLF) [30], where

$$c_1 \|x\|^2 \leq V(x) \leq c_2 \|x\|^2 \quad \text{and} \quad \inf_{u \in \mathbb{U}} \left[\dot{V}(x,u) + c_3 V(x) \right] \leq 0,$$

with positive constants c_1, c_2, c_3. This implies exponential convergence of trajectories to the desired equilibrium. In practice, such CLFs can be employed in *optimization-based* or *min-norm* controllers, subject to input constraints. These ideas naturally dovetail with the Quadratic Program (QP) formulations that will appear later (see Section 2.4), thereby enabling real-time feasible controller synthesis [31, 29, 30].

2.2 Barrier Functions

In this section, we introduce two common types of barrier functions that enforce set invariance in continuous-time systems without explicit input ($\dot{x} = f(x)$). Later, in Section 2.3, we will incorporate control inputs to obtain *control barrier functions*. As discussed in the **Safe Set** definition (see (2.8)), the overarching goal is to keep the state within a desired safe region $\mathcal{C}$. We focus here on *reciprocal barrier functions (RBF)* and *zeroing barrier functions (ZBF)*, following the treatments in [7, 10] and related works.

2.2.1 Reciprocal BF

Consider a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$ whose zero-level set defines the boundary $\partial\mathcal{C}$. We say that the *interior* of $\mathcal{C}$ is $\{x \mid h(x) > 0\}$ and aim to ensure forward invariance of that interior under $\dot{x} = f(x)$. A *reciprocal barrier function* $B(x)$ is one that tends to *infinity* as the state x approaches $\partial\mathcal{C}$. For example, a typical choice is (2.12):

$$B(x) = \frac{1}{h(x)},$$

so that $B(x) \to \infty$ whenever $h(x) \to 0$. More generally, one can define B satisfying

$$B(x) \to \infty \quad \text{as } x \to \partial\mathcal{C},$$

and impose a suitable differential inequality (involving $\dot{B}(x)$) so that solutions cannot reach $\partial\mathcal{C}$ in finite time. In essence, if $\dot{B}$ is constrained to grow no faster than a function of B itself, then the trajectory stays in the safe interior indefinitely [32, 33, 34].

In the control setting, such reciprocal-type barrier conditions appear in Section 2.3.1 (Definitions 2.3.3–2.12), where they enforce forward invariance of $\mathrm{Int}(\mathcal{C})$. Absent a control input, the same principle applies: if $B(x(t))$ remains finite for all t, then the state $x(t)$ remains in $\mathrm{Int}(\mathcal{C})$.

2.2.2 Zeroing BF

An alternative approach is to require that the chosen barrier function *vanish* on the boundary of $\mathcal{C}$. That is, define $h : \mathbb{R}^n \to \mathbb{R}$ with

$$h(x) = 0 \quad \text{on } \partial\mathcal{C}, \quad h(x) > 0 \text{ for } x \in \text{Int}(\mathcal{C}), \tag{2.4}$$

and consider h itself as a *zeroing barrier function*. In this case, the derivative $\dot{h}(x)$ is constrained to ensure that the set $\{x \mid h(x) \geq 0\}$ cannot be exited. Concretely, one often imposes a condition such as

$$L_f\, h(x) \;\geq\; -\alpha\Big(h(x)\Big), \tag{2.5}$$

where α is an *extended class* $\mathcal{K}$ function (i.e., strictly increasing, $\alpha(0) = 0$). This implies that $h(x)$ cannot cross zero from above, thus preserving $\{h(x) \geq 0\}$ [33, 35].

Zeroing barrier functions do not blow up near the boundary (as reciprocal barriers do); instead, they remain *zero* exactly on $\partial\mathcal{C}$. For real-time implementations, this can be beneficial, since numerical algorithms need not deal with infinite function values [35]. Moreover, in certain circumstances, a ZBF can induce an asymptotically stable set, yielding stronger invariance properties if $\mathcal{C}$ is compact.

2.2.3 Relationships between RBF and ZBF

Both reciprocal and zeroing barrier functions yield *sufficient* conditions for forward invariance. Specifically, if an RBF B with $B \to \infty$ on $\partial\mathcal{C}$ satisfies the appropriate growth constraint on $\dot{B}$, then $\text{Int}(\mathcal{C})$ is invariant. Likewise, if a ZBF h meets its derivative condition on $\partial\mathcal{C}$, the set $\mathcal{C} = \{x \mid h(x) \geq 0\}$ is invariant. In certain *contractive* scenarios—where the flow on $\partial\mathcal{C}$ points strictly inward—one can also establish that the existence of a RBF or a ZBF is *necessary* for invariance (at least under compactness assumptions [36]).

In practice, the choice between reciprocal versus zeroing barrier functions depends on convenience, numerical conditioning, and the desired boundary behavior. Section 2.3.1 (and specifically (2.12)–(2.19)) provides the extension of reciprocal barriers to the control-affine case, while zeroing barrier methods similarly generalize to *control zeroing barrier functions* (ZCBFs) under mild assumptions (not elaborated here). In both approaches, ensuring $\dot{B}$ (resp. $\dot{h}$) meets the barrier condition is the key to safety.

2.3 Control Barrier Functions

While barrier functions (see Section 2.2) verify set invariance *in the absence* of a control input, they do not directly provide a method to synthesize a control law that *enforces* invariance. In analogy with how Lyapunov functions were extended to control Lyapunov functions [29], we now define *control barrier functions (CBFs)* that incorporate the input u. This includes extending the reciprocal barrier function condition and the zeroing barrier function condition [7] to allow for a larger set of feasible controls. As we will see in Section 2.4, the usefulness of CBFs becomes most apparent when they are combined with control Lyapunov functions in a real-time quadratic program.

2.3.1 Reciprocal CBFs

In this section, Control Barrier Functions (CBFs) and their importance for real-time safety-critical control are formally introduced. Consider a continuous-time deterministic control system with state $x(t) \in D \subseteq \mathbb{R}^n$ and control input $u(t) \in \mathbb{U} \subseteq \mathbb{R}^m$ for $t \geq 0$. The state dynamics are described by the ordinary differential equation

$$\dot{x}(t) = f(x(t)) + g(x(t))u(t), \tag{2.6}$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ are continuously differentiable functions, and $u(t) \in \mathbb{R}^m$ is the control input. Moreover, given a Lipschitz continuous control law $u = k(x)$, the closed-loop system is expressed as

$$\dot{x}(t) = f(x(t)) + g(x(t))k(x(t)), \tag{2.7}$$

For any initial condition $x(0) = x_0$, the unique solution is denoted by $\phi(t, x_0)$ for t in the maximal interval of existence J.

Definition 2.3.1 (Safe Set). A set $\text{Int}(\mathcal{C}) \subseteq \mathbb{R}^n$ is said to be a *safe set* for the closed-loop system under the control law $u = k(x)$ if for every initial state $x(0) \in \text{Int}(\mathcal{C})$, the unique solution $\phi(t, x(0))$ satisfies

$$\phi(t, x(0)) \in \text{Int}(\mathcal{C}) \quad \text{for all } t \geq 0. \tag{2.8}$$

This property ensures that once the state is within $\mathcal{C}$, it remains safe for all future times. The distance of a point $x \in \mathbb{R}^n$ to a set $\mathcal{C} \subseteq \mathbb{R}^n$ is defined as $\|x\|_{\mathcal{C}} := \inf_{y\in\mathcal{C}} \|x-y\|$.

In line with the definition of a safe set above, the interior of $\mathcal{C}$ is equivalently represented as the super-level set of a continuously differentiable function $h : D \to \mathbb{R}$. That

is, define

$$\mathcal{C} = \{x \in D \subset \mathbb{R}^n : h(x) \geq 0\}, \tag{2.9}$$

$$\partial\mathcal{C} = \{x \in D \subset \mathbb{R}^n : h(x) = 0\}, \tag{2.10}$$

$$\text{Int}(\mathcal{C}) = \{x \in D \subset \mathbb{R}^n : h(x) > 0\}. \tag{2.11}$$

It is assumed that $\text{Int}(\mathcal{C}) \neq \emptyset$ and that $\mathcal{C}$ has no isolated points, i.e., $\overline{\text{Int}(\mathcal{C})} = \mathcal{C}$. The reciprocal barrier function is then defined as

$$B(x) = \frac{1}{h(x)}. \tag{2.12}$$

The controller $k(x)$ is verified to ensure safety by utilizing Reciprocal Control Barrier Functions (RCBFs), which are defined as follows.

Definition 2.3.2 (**Reciprocal Control Barrier Function**)**.** A continuously differentiable function $B : \text{Int}(\mathcal{C}) \to \mathbb{R}$ is called a *reciprocal control barrier function (RCBF)* if it is defined by (2.12) and satisfies:

1) **Barrier Property:** $B(x)$ is finite for all $x \in \text{Int}(\mathcal{C})$. Moreover, since $h(x) \to 0$ as $x \to \partial\mathcal{C}$,

$$B(x) = \frac{1}{h(x)} \to \infty \quad \text{as } x \to \partial\mathcal{C}, \tag{2.13}$$

ensuring that unsafe conditions (i.e., approaching the boundary of $\mathcal{C}$) are heavily penalized.

2) **Descent Along Trajectories:** The time derivative along trajectories,

$$\dot{B}(x) := \lim_{t \to 0^+} \frac{B(\phi(t, x)) - B(x)}{t}, \tag{2.14}$$

is well-defined and satisfies

$$\dot{B}(x) = -\Phi(x), \tag{2.15}$$

where $\Phi : D \to \mathbb{R}$ is continuous and positive.

Definition 2.3.3 (Safety Conditions for RCBF)**.** Let B be an RCBF as in the previous definition. Then, B is said to satisfy the *safety conditions* if there exist class $\mathcal{K}$ functions K_1, K_2, and K_3 such that for all $x \in \text{Int}(\mathcal{C})$:

1) **Boundedness Condition:** The barrier function $B(x)$ is bounded above and below

by class $\mathcal{K}$ functions of $h(x)$ as follows:

$$\frac{1}{K_1(h(x))} \leq B(x) \leq \frac{1}{K_2(h(x))}. \tag{2.16}$$

2) **Safety Condition:** Enforcing safety requires that the following inequality is satisfied:

$$\inf_{u\in\mathbb{U}}\Big[L_fB(x) + L_gB(x)u\Big] \leq K_3\Big(\frac{1}{B(x)}\Big), \tag{2.17}$$

where the Lie derivatives are defined as

$$L_fB(x) = \nabla B(x)^\top f(x), \quad L_gB(x) = \nabla B(x)^\top g(x). \tag{2.18}$$

3) **Control Set Definition:** Accordingly, the set of admissible control inputs is defined as

$$K_{\text{rcbf}}(x) = \{u \in \mathbb{U} \,|\, L_fB(x) + L_gB(x)u \leq K_3(h(x))\}\,. \tag{2.19}$$

If a controller selects inputs u from $K_{\text{rcbf}}(x)$ for all $x \in \text{Int}(\mathcal{C})$, then the safe set is forward invariant [7, 10]. The RCBF B is said to be locally Lipschitz continuous if K_3 and $\frac{\partial B}{\partial x}$ are both locally Lipschitz continuous.

2.3.2 Zeroing CBFs and Higher Relative Degree

In addition to reciprocal-type barrier functions, an alternative *zeroing barrier function* can be extended to the control-affine case to form a *zeroing control barrier function* (ZCBF). Concretely, if h is the ZBF introduced in Section 2.2.1, then the *ZCBF condition* is often stated in the form

$$\sup_{u\in\mathbb{U}}\Big[\,L_fh(x) \;+\; L_gh(x)\,u \;+\; \alpha\Big(h(x)\Big)\Big] \;\geq\; 0, \tag{2.20}$$

where α is an extended class $\mathcal{K}$ function. Defining

$$K_{\text{zcbf}}(x) \;=\; \Big\{\,u \in \mathbb{U}\,\Big|\,L_fh(x) \;+\; L_gh(x)\,u \;+\; \alpha\Big(h(x)\Big) \;\geq 0\Big\}, \tag{2.21}$$

it follows that any locally Lipschitz controller u taking values in $K_{\text{zcbf}}(x)$ renders $\mathcal{C} = \{x : h(x) \geq 0\}$ forward invariant [7].

Higher Relative Degrees. When the function h (or equivalently $B(x)$) has relative degree greater than one with respect to the dynamics $\dot{x} = f(x) + g(x)u$, the standard forms of RCBF or ZCBF yield trivial constraints (e.g., $L_g h = 0$). One can address this by constructing *higher relative degree* CBFs through additional terms (e.g., backstepping-like techniques) or by combining derivatives of h in a way that yields a valid barrier function. For instance, one approach is to define

$$B_r(x) \;=\; \frac{1}{h(x)} \;+\; H\Big(L_f^{r-1} h(x)\Big), \tag{2.22}$$

where $r \geq 2$ is the relative degree of h, and H is a suitably chosen function ensuring positivity and differentiability [37]. Detailing these constructions lies beyond our immediate scope, but we note that they are crucial for systems where the barrier function must be differentiated multiple times before the input u appears.

2.4 Optimization-based Control using QP

In this section, we address the fundamental question of how to synthesize controllers that simultaneously respect safety and achieve stability or performance objectives. Drawing inspiration from the *exponentially stabilizing control Lyapunov function (ES-CLF)* approach, we show how both control Lyapunov functions (CLFs) and control barrier functions (CBFs) can be incorporated into a single real-time quadratic program (QP). This framework balances safety (via CBF constraints) with stability or other performance considerations (via CLF conditions).

2.4.1 CLF QP

Recall from Section 2.1 that a function $V(x)$ is called an *exponentially stabilizing control Lyapunov function (ES-CLF)* if there exist positive constants c_1, c_2, c_3 such that for all x,

$$c_1 \|x_1\|^2 \;\leq\; V(x) \;\leq\; c_2 \|x_1\|^2, \tag{2.23}$$

and

$$\inf_{u \in \mathbb{U}} \Big[L_f V(x) + L_g V(x)\, u + c_3\, V(x) \Big] \;\leq\; 0. \tag{2.24}$$

These conditions ensure that there is a family of inputs $u \in K_{\text{clf}}(x)$ stabilizing the system exponentially to the zero dynamics (see [30, 31, 29] for additional details). When $\mathbb{U} = \mathbb{R}^m$, one obtains a *min-norm* controller by solving a *quadratic program*. Concretely, a *CLF*

QP may be formulated as

$$\begin{aligned} u^*(x) \;&= \underset{u\in\mathbb{R}^m}{\operatorname{argmin}} \; \frac{1}{2}\|u\|^2 \\ &\text{s.t. } L_f V(x) + L_g V(x)\, u \;\leq\; -c_3\, V(x). \end{aligned} \tag{2.25}$$

Additional linear constraints on u (such as actuation limits) can also be included in the QP as needed. In real-time applications, this approach has successfully achieved fast, reliable stabilizing control [31, 6].

2.4.2 CBF QP

In parallel to the CLF QP, we can formulate a *CBF QP* that enforces the barrier condition (i.e., safety) in a minimally invasive manner. For instance, one could similarly design a QP for zeroing control barrier functions (ZCBFs) by imposing $\sup_{u\in\mathbb{U}}[\, L_f h(x) + L_g h(x)\, u + \alpha(h(x))] \geq 0$. However, for this thesis, we focus on *reciprocal control barrier functions (RCBFs)*; the resulting QP is summarized below.

Having established the barrier condition in (2.17), a control policy is designed that minimally modifies a nominal control input $u_{ref}(x)$ to ensure safety. In particular, the nominal controller is adjusted through a Quadratic Program (QP) that enforces the barrier condition as a constraint. The QP formulation is given by

$$\begin{aligned} u^*(x) = \underset{u\in\mathbb{U}\subseteq\mathbb{R}^m}{\operatorname{argmin}} \; &\|u - u_{ref}(x)\|^2 \\ \text{s.t.} \;\; &L_f B(x) + L_g B(x)\, u \;-\; \kappa\, h(x) \;\leq\; 0, \end{aligned} \tag{2.26}$$

where $\mathbb{U} \subseteq \mathbb{R}^m$ denotes the set of admissible control inputs and κ is a positive constant (by choosing $K_3(s) = \kappa\, s$).

Alternatively, when the safety condition (2.17) is violated, a closed-form minimal-norm adjustment can be computed as

$$\Delta u = \begin{cases} -\dfrac{L_f B(x) + L_g B(x) u_{ref}(x) - \kappa\, h(x)}{\|L_g B(x)\|^2 + \varepsilon}\, L_g B(x), & \text{if } L_f B(x) + L_g B(x) u_{ref}(x) > \kappa\, h(x), \\ 0, & \text{otherwise,} \end{cases} \tag{2.27}$$

where $\varepsilon > 0$ is a small regularization constant ensuring the expression is well-defined. Consequently, the final control law is defined as

$$u(x) = u_{ref}(x) + \Delta u. \tag{2.28}$$

This controller modifies $u_{ref}(x)$ only when necessary to ensure that the condition (2.17) holds for all $x \in \mathrm{Int}(\mathcal{C})$. Under standard assumptions, this implies that the safe set remains forward invariant.

2.4.3 CLF-CBF QP

A key benefit of the QP perspective is that it provides a principled way to *unify* performance/stability (via CLFs) with safety constraints (via CBFs) in a *single* optimization problem. For instance, one might impose:

$$\begin{aligned}(u^*, \delta^*)(x) \;=\; \underset{(u,\delta)\in\mathbb{R}^{m+1}}{\operatorname{argmin}} \;& \tfrac{1}{2}\,\|u\|^2 \;+\; p\,\delta^2 \\ \text{s.t. } & L_f V(x) + L_g V(x)\,u \;\leq\; -\,c_3\,V(x) \;+\; \delta, \\ & L_f B(x) + L_g B(x)\,u \;\leq\; \alpha\Big(h(x)\Big),\end{aligned} \tag{2.29}$$

where δ is a slack variable penalized by $p > 0$. This ensures feasibility in the event that strict enforcement of the CLF condition might conflict with the barrier constraint. By adjusting the weight p, one can trade off stability performance versus safety. More sophisticated formulations (e.g., [7, 31]) provide Lipschitz continuity of the resulting controller, ensuring real-time implementability.

In this work, we do not dive further into the combined CLF-CBF QP, as our emphasis is on employing *standalone* CLF or CBF constraints for separate objectives. Nevertheless, the unified approach (2.29) highlights the flexibility of optimization-based control to simultaneously satisfy multiple control objectives.

Chapter 3

Problem Formulation

3.1 Lyapunov's Characterization for Domain of Attraction

Consider an autonomous system

$$\dot{x}(t) \;=\; f(x(t)), \quad x \in \mathbb{R}^n, \tag{3.1}$$

and let $\phi(t,x)$ be the unique solution of (3.1) with initial condition x. We denote by $\mathcal{D}(\mathcal{A})$ the *domain of attraction* of a compact set $\mathcal{A} \subset \mathbb{R}^n$, namely $\mathcal{D}(\mathcal{A}) \;=\; \Big\{ x \in \mathbb{R}^n \;\Big|\; \phi(t,x)$ remains in a neighborhood of $\mathcal{A}$ for all $t \geq 0$ and $\phi(t,x) \to \mathcal{A}$ as $t \to \infty \Big\}$. We assume that $\mathcal{D}(\mathcal{A})$ is an open set.

The following result states that if $\mathcal{D}(\mathcal{A})$ is covered by a *maximal Lyapunov function*, then no strictly larger open set can also be forward-invariant and still converge to $\mathcal{A}$.

Theorem 3.1.1 (Maximal Lyapunov Function)**.** [26] Let $D \subseteq \mathbb{R}^n$ be an open set. Suppose there exists a continuous function $V : D \to \mathbb{R}$ such that $\mathcal{A} \subseteq D$ and the following conditions hold:

(1) V is *positive definite* on D with respect to $\mathcal{A}$, i.e., $V(x) \;=\; 0$ for all $x \in \mathcal{A}$, and $V(x) > 0$ for all $x \in D \setminus \mathcal{A}$.

(2) The derivative of V along solutions of (3.1) is well defined for all $x \in D$, via

$$\dot{V}(x) \;:=\; \lim_{t \to 0^+} \frac{V(\phi(t,x)) - V(x)}{t},$$

and there is a continuous function $\Phi : D \to \mathbb{R}$, which is positive definite with

respect to $\mathcal{A}$, such that

$$\dot{V}(x) \;=\; -\,\Phi(x) \quad \forall\, x \in D. \tag{3.2}$$

(3) $V(x) \to \infty$ as x approaches ∂D or as $\|x\| \to \infty$ within D.

Then $D = \mathcal{D}(\mathcal{A})$.

Moreover, if f is sufficiently smooth (e.g. Lipschitz) on $\mathcal{D}(\mathcal{A})$, one can often choose V to be continuously differentiable on D. Under mild hypotheses on f, the existence of such a *maximal Lyapunov function* is both necessary and sufficient for characterizing $\mathcal{D}(\mathcal{A})$.

Remark 3.1.2. The above statement highlights that any open set D satisfying the three properties must coincide with the entire domain of attraction $\mathcal{D}(\mathcal{A})$. Thus, a *maximal Lyapunov function* is a single, continuous scalar function that *blows up* on the boundary of $\mathcal{D}(\mathcal{A})$ and decays monotonically along trajectories to zero on $\mathcal{A}$.

3.2 Zubov's Characterization for Domains of Attraction

An alternative but equivalent way to capture $\mathcal{D}(\mathcal{A})$ is via the classical *Zubov method*, originally stated for autonomous systems with a single equilibrium and subsequently generalized to more general attractors. The next theorem uses a function W taking values in $[0, 1)$ on $\mathcal{D}(\mathcal{A})$, extended to take the value 1 on the complement.

Theorem 3.2.1 (Zubov's Theorem)**.** [26, 38] Let $D \subseteq \mathbb{R}^n$ be an open set containing $\mathcal{A}$. Then $D = \mathcal{D}(\mathcal{A}) \iff$ there exist continuous functions $W : D \to \mathbb{R}$, $\Psi : D \to \mathbb{R}$ such that:

(1) $0 < W(x) < 1$ for all $x \in D \setminus \mathcal{A}$, and $W(x) = 0$ for all $x \in \mathcal{A}$.

(2) Ψ is positive definite on D with respect to $\mathcal{A}$.

(3) For any sufficiently small $c_3 > 0$, there exist c_1 and c_2 such that

$$\|x\|_{\mathcal{A}} \;\geq\; c_3 \quad\Longrightarrow\quad W(x) > c_1, \quad \Psi(x) > c_2.$$

(4) $W(x) \to 1$ as $x \to y$ for any $y \in \partial D$ (or as $\|x\| \to \infty$ within D).

(5) Along trajectories of (3.1), the function W satisfies

$$\dot{W}(x) \;=\; -\,\Psi(x)\,\Big(1 - W(x)\Big), \tag{3.3}$$

where $\dot{W}(x)$ denotes the derivative of W along solutions, similarly to (3.2).

Remark 3.2.2. One sees that (3.3) is a *Zubov-type functional equation*. In particular, $\Psi(x)$ acts as a *rate* of exponential decay on the quantity $1 - W(x)$. Since W vanishes exactly on $\mathcal{A}$ and tends to 1 on ∂D, its strict sublevel set

$$\{\, x \in D \mid W(x) < 1 \,\} \;=\; D$$

is precisely the domain of attraction.

Remark 3.2.3. It is immediate from (3.3) that if one sets

$$V(x) \;=\; -\,\tfrac{1}{\alpha}\,\ln\Big(1 - W(x)\Big) \quad \text{for some } \alpha > 0,$$

then $\dot{V}(x) = -\Phi(x)$ for $\Phi(x) = \alpha\,\Psi(x)$. Conversely, if a function V satisfies Theorem 3.1.1, then

$$W(x) \;=\; 1 - \exp\Big(-\alpha\,V(x)\Big)$$

satisfies Zubov's theorem. This connection between the two connections is further explored, and in fact proved to be equivalent in the next section.

3.3 Equivalence of Characterizations & Benefits

We now provide a proof (sketch) of the equivalence between the maximal Lyapunov characterization (Theorem 3.1.1) and Zubov's characterization (Theorem 3.2.1). We do so by adapting the classic Zubov construction via a *cost function* and associated *value function*, as outlined in [38]. Readers are recommended to refer to the original work to work out all the details of the proof, but the major essence is captured in the following sections:

Assumptions (H0)–(H6) for the Autonomous System

Throughout, assume $f : \mathbb{R}^n \to \mathbb{R}^n$ has the following properties:

(H0) **Local Lipschitz Continuity.** There is a function $\gamma \in \mathcal{K}_\infty$ (i.e. γ continuous,

strictly increasing, $\gamma(0) = 0$) such that for any $R > 0$ there is a $C_R > 0$ with

$$\|f(x) - f(y)\| \;\le\; C_R\Big(1 + \gamma(\|x\| + \|y\|)\Big)\,\|x - y\|, \quad \|x\|, \|y\| \le R. \tag{3.4}$$

This ensures uniqueness of solutions and controlled growth.

(H1) **Equilibria in $\mathcal{A}$.** For all $x \in \mathcal{A}$, $f(x) = 0$.

(H2) **Local Asymptotic Stability.** There is $r > 0$ such that any solution $\phi(\cdot, x)$ with $\|x\|_{\mathcal{A}} < r$ remains in a small neighborhood of $\mathcal{A}$ and converges to $\mathcal{A}$ as $t \to \infty$.

(H3) **Running Cost g.** A continuous function $g : \mathbb{R}^n \to \mathbb{R}$ is chosen, satisfying a similar Lipschitz-type property (possibly with the same or a larger γ) and such that $g(x)$ is positive outside any neighborhood of $\mathcal{A}$. Formally, for every $c > 0$, there is a uniform $g_c > 0$ with $g(x) \ge g_c$ for $\|x\|_{\mathcal{A}} \ge c$.

(H4) **Integrability near $\mathcal{A}$.** On the local set $\|x\|_{\mathcal{A}} \le r$, $g(x)$ remains bounded in such a way that the integral $\int_0^\infty g(\phi(t, x))\, dt$ converges for all x with $\|x\|_{\mathcal{A}} \le r$.

(H5) **Large Cost Far from $\mathcal{A}$.** Outside large compact sets around $\mathcal{A}$, g dominates $\|f(x)\|$ and grows in such a manner that solutions approaching the boundary ∂D give an unbounded integral cost.

(H6) **Unboundedness w.r.t. f.** As $\|x\| \to \infty$, the ratio $g(x)/(1+\|f(x)\|)$ also becomes large, ensuring that the "maximal solution" argument used below for uniqueness can be applied.

Construction of the Value Function and Zubov Function

To prove Theorem 3.2.1 from Theorem 3.1.1 and vice versa, one introduces:

Definition 3.3.1 (Cost Functional and Value Function)**.** For $x \in \mathbb{R}^n$, define the *cost functional*

$$J(x) \;:=\; \int_0^\infty g(\phi(t, x))\, dt,$$

provided $\phi(\cdot, x)$ does not blow up in finite time. If a trajectory exits a prescribed open set D (where we want the Zubov function to live), set $J(x) = +\infty$. Then define

$$V(x) \;:=\; J(x).$$

By assumptions (H3)–(H5), every $x \in \mathcal{D}(\mathcal{A}) \cap D$ yields a convergent integral $J(x) < \infty$. Conversely, if a point $x \notin \mathcal{D}(\mathcal{A})$ or if $\phi(\cdot, x)$ reaches ∂D, the hypotheses imply $J(x) = +\infty$. This ensures V is finite precisely on $\mathcal{D}(\mathcal{A}) \cap D$.

Definition 3.3.2 (Zubov Function)**.** Choose any constant $\alpha > 0$. Define

$$W(x) \;:=\; 1 \;-\; \exp\Big(-\alpha\, V(x)\Big), \quad x \in D. \tag{3.5}$$

When $V(x) = +\infty$, we interpret $\exp(-\alpha\, V(x)) = 0$, so $W(x) = 1$ there.

Notice that $\mathcal{A} \subseteq D$ and $V(x) = 0$ exactly when $x \in \mathcal{A}$, so $W(x) = 0$ on $\mathcal{A}$. Also, $W(x) \in [0, 1)$ if and only if $V(x) \in [0, \infty)$, i.e., $x \in \mathcal{D}(\mathcal{A}) \cap D$. On ∂D or outside $\mathcal{D}(\mathcal{A})$, one has $W(x) = 1$. Hence, the set

$$\Big\{\, x \in D : W(x) < 1 \,\Big\} \;=\; \mathcal{D}(\mathcal{A}) \cap D.$$

As we shall see, if $D \supseteq \mathcal{D}(\mathcal{A})$ was arbitrary, the "Zubov solution" forces D itself to be the entire domain of attraction. We next show W satisfies the required derivative condition (3.3).

Verification of the Zubov Equation and Proof of Equivalence

Step 1: Derivative along trajectories. By direct computation and the chain rule, for $x \in D \cap \mathcal{D}(\mathcal{A})$ where V is finite,

$$\dot{V}(x) \;=\; \lim_{t \to 0^+} \frac{V(\phi(t,x)) - V(x)}{t} \;=\; \lim_{t \to 0^+} \frac{1}{t} \int_0^t \Big[g\Big(\phi(\tau, x)\Big) \Big] d\tau \;=\; g(x).$$

Hence,

$$\dot{W}(x) \;=\; \nabla W(x) \cdot f(x) \;=\; -\,\alpha\, \exp\Big(-\alpha\, V(x)\Big)\, \dot{V}(x) \;=\; -\,\alpha\, \exp\Big(-\alpha\, V(x)\Big)\, g(x).$$

But since $\exp\Big(-\alpha\, V(x)\Big) = 1 - W(x)$, we obtain

$$\dot{W}(x) \;=\; -\,\alpha \Big(1 - W(x)\Big)\, g(x).$$

Hence, if we define $\Psi(x) := \alpha\, g(x)$, which is positive definite with respect to $\mathcal{A}$, we arrive at

$$\dot{W}(x) \;=\; -\,\Psi(x) \Big(1 - W(x)\Big),$$

which is precisely the Zubov equation (3.3) in Theorem 3.2.1.

Step 2: Boundary Behavior and Uniqueness. From assumptions (H5) and (H6), as x approaches ∂D or "goes to infinity" in D, the integral $\int_0^\infty g(\phi(t,x))\, dt$ blows up to $+\infty$. This forces $V(x) \to \infty$ and thus $W(x) \to 1$. A standard sub-/super-solution comparison argument (or a strict monotonicity argument) shows that if any other pair $\widetilde{W}, \widetilde{\Psi}$

attempted to satisfy the same boundary and positivity conditions, it must coincide with $\{W, \Psi\}$. Hence, uniqueness of the solution to the Zubov equation (up to the transform by a positive factor α) is secured.

Step 3: Proof of Theorem 3.2.1 $\Longrightarrow$ Theorem 3.1.1. Suppose W and Ψ satisfy all items of Theorem 3.2.1. Define $V(x)$ by

$$V(x) \;=\; -\frac{1}{\alpha}\,\ln\Big(1 - W(x)\Big),$$

where α is a positive constant. Then

$$\dot{V}(x) \;=\; -\,\frac{1}{\alpha(1 - W(x))}\,\dot{W}(x) \;=\; \frac{\Psi(x)}{\alpha} \;=\; \Phi(x).$$

In particular, Φ is positive definite with respect to $\mathcal{A}$. Because $W \to 1$ on ∂D or for $\|x\| \to \infty$ in D, we have $V \to \infty$, satisfying condition (3) in Theorem 3.1.1. Also $V = 0$ on $\mathcal{A}$. Thus V satisfies Theorem 3.1.1 exactly on D, forcing $D \subseteq \mathcal{D}(\mathcal{A})$. But also by assumption we must have $D \supseteq \mathcal{D}(\mathcal{A})$. Therefore $D = \mathcal{D}(\mathcal{A})$.

Step 4: Proof of Theorem 3.1.1 $\Longrightarrow$ Theorem 3.2.1. Given V from Theorem 3.1.1, define $\alpha > 0$ and let

$$W(x) \;=\; 1 \;-\; \exp\Big(-\,\alpha\,V(x)\Big).$$

Then we compute $\dot{W}(x) = -\alpha\,\exp(-\alpha\,V(x))\,\dot{V}(x)$. Since $\dot{V}(x) = -\Phi(x)$ with Φ positive definite, it follows that

$$\dot{W}(x) \;=\; \alpha\,\exp\Big(-\alpha\,V(x)\Big)\,\Phi(x) \;=\; -\Big[-\alpha\,\exp(-\alpha\,V(x))\,\Phi(x)\Big].$$

But $\exp(-\alpha\,V(x)) = 1 - W(x)$. Hence if we define $\Psi(x) = \alpha\,\Phi(x)$, we get

$$\dot{W}(x) \;=\; -\,\Psi(x)\,\Big[\,1 - W(x)\Big].$$

Moreover, Ψ is continuous and positive definite with respect to $\mathcal{A}$. The boundary and positivity conditions likewise hold because $V \to \infty$ forces $W \to 1$. Thus $\{\,W, \Psi\}$ meets all items of Theorem 3.2.1 exactly on D. Once again, no strictly larger open set can satisfy these properties, so $D = \mathcal{D}(\mathcal{A})$.

We have shown how one passes from a $\dot{V} = -\Phi$ Lyapunov function (Theorem 3.1.1) to a $\dot{W} = -\,\Psi(\,\cdot\,)\Big(1 - W(\,\cdot\,)\Big)$ Zubov function (Theorem 3.2.1), and back again, with a one-to-one correspondence. All assumptions (H0)–(H6) ensure regularity, positivity, boundary

blow-up, and uniqueness so that the set $\{W < 1\} = \{V < \infty\}$ exactly captures $\mathcal{D}(\mathcal{A})$. Hence, *Theorem 3.1.1 and Theorem 3.2.1 are equivalent characterizations of the domain of attraction.*

However, characterization using Zubov's Theorem has distinct advantages because the result is bounded, such as overcoming numerical problems during synthesis, improving training stability, and allowing user-adjustable sub-level sets according to control requirements. Works such as [26] and [27] have shown that maximal domains of attraction can be recovered, which closely approximate analytical solutions, and that the learned Lyapunov functions provide benefits during the verification process as well. Motivated by these successes and the similarities in structure between Lyapunov functions and Reciprocal Barrier Functions [39], the work in this thesis was carried out to extend these benefits to the realm of robot safety, as well.

3.4 Zubov's Characterization for Safe Regions

As described in Section 1, we now present an equivalent Zubov-type theorem for safe regions, drawing parallels from the work on Lyapunov functions and domains of attraction [28, 26].

Define a function $W_N : D \to \mathbb{R}$ that is related to B via a scalar transformation (made explicit later in (3.9), (3.12)). The following result holds.

Theorem 3.4.1 (Zubov's Characterization for Safe Regions)**.** *Let $D \subseteq \mathbb{R}^n$ be an open set containing $\mathcal{C}$. Then, the forward invariant safe set is characterized by*

$$\text{Int}(\mathcal{C}) = \{x \in D : W_N(x) < 1\}, \tag{3.6}$$

if and only if there exist continuous functions $W_N : D \to \mathbb{R}$ and $\Psi : D \to \mathbb{R}$ such that:

1. *$W_N(x) < 1 \,\forall\, x \in \text{Int}(\mathcal{C})$, $W_N(x) \to 1$ as $x \to \partial\mathcal{C}$ and $W_N(x) = 1 \,\forall\, x \in D \setminus \mathcal{C}$.*
2. *Ψ is positive definite on D with respect to $\mathcal{C}$.*
3. *For any sufficiently small $c_3 > 0$, there exist constants $c_1, c_2 > 0$ such that $\|x\|_{\mathcal{C}} > c_3 \implies W_N(x) > c_1$ and $\Psi(x) > c_2$.*
4. *W_N and Ψ satisfy the following PDE:*

$$\dot{W}_N(x) = -\Psi(x)\Big(1 - W_N(x)\Big), \tag{3.7}$$

where $\dot{W}_N(x)$ denotes the derivative of W_N along trajectories of (2.6).

This theorem is closely related to the satisfaction of the conditions in Section 2.3.1, and follows directly from Zubov's theorem [40] for domains of attraction (characterized by sub-level sets of Lyapunov functions). By a reciprocal relationship (see [39]), a Lyapunov function can be transformed into an RBF, thereby preserving the forward-invariant sub-level set structure, which defines the safe set in this case. An explicit proof follows in the same way as shown in Section 3.3.

3.5 Construction of W_N

Let $\beta : [0, \infty) \to \mathbb{R}$ satisfy:

$$\dot{\beta}(s) = (1 - \beta(s))\, \psi(\beta(s)), \quad \beta(0) = 0, \tag{3.8}$$

where $\psi : [0, \infty) \to \mathbb{R}$ is locally Lipschitz with $\psi(s) > 0$ for all $s > 0$. (Then β is continuously differentiable, strictly increasing, with $\beta(s) \to 1$ as $s \to \infty$.)

Assume that $B : \mathbb{R}^n \to [0, \infty)$ is an RCBF candidate for $\mathrm{Int}(\mathcal{C})$. Define the transformed function $W_N : \mathbb{R}^n \to \mathbb{R}$ as:

$$W_N(x) = \begin{cases} \beta\Big(B(x)\Big), & \text{if } B(x) < \infty, \\ 1, & \text{otherwise.} \end{cases} \tag{3.9}$$

By construction, we have that $W_N(x) < 1$ for $x \in \mathrm{Int}(\mathcal{C})$ and $W_N(x) \to 1$ as $B(x) \to \infty$ (i.e., as x approaches $\partial\mathcal{C}$).

To study the evolution of W_N along trajectories of (2.6), we already know (2.15) from Section 2.3.1. Then, by the chain rule,

$$\dot{W}_N(x) = \dot{\beta}\Big(B(x)\Big)\, \dot{B}(x) = -\dot{\beta}\Big(B(x)\Big)\, \Phi(x). \tag{3.10}$$

A particularly useful choice is to take $\psi(s) = \alpha(1 + s)$ for some constant $\alpha > 0$. In this case, (3.8) becomes

$$\dot{\beta}(s) = \alpha\,(1 - \beta(s))(1 + \beta(s)), \quad \beta(0) = 0. \tag{3.11}$$

It can be verified that the unique solution is $\beta(s) = \tanh(\alpha s)$. Thus, for $B(x) < \infty$ we have

$$W_N(x) = \tanh\Big(\alpha\, B(x)\Big). \tag{3.12}$$

Differentiating along trajectories of (2.6) and using (2.15), it follows that:

$$\begin{aligned}\dot{W}_N(x) &= \alpha\Big[1-\tanh^2\big(\alpha\, B(x)\big)\Big]\dot{B}(x)\\ &= \alpha\,(1-W_N(x))(1+W_N(x))\,\dot{B}(x)\\ &= -\alpha\,(1-W_N(x))(1+W_N(x))\,\Phi(x).\end{aligned} \tag{3.13}$$

Thus, comparing with (3.7), we have:

$$\Psi(x) = \alpha(1+W_N(x))\,\Phi(x). \tag{3.14}$$

Remark 3.5.1. Two common choices for the transformation arise from different selections of $\psi(s)$:

(i) If $\psi(s) = \alpha$ (a constant), then $\beta(s) = 1-\exp(-\alpha s)$ is the solution of:

$$\dot{\beta}(s) = \alpha(1-\beta(s)), \quad \beta(0) = 0.$$

(ii) For the numerical examples in this paper, we choose $\Phi(x) = ||x||_{\mathcal{C}}^2$ and $\psi(s) = \alpha(1+s)$, then, as shown above, $\beta(s) = \tanh(\alpha s)$.

In either case, the transformation yields a bounded function W_N that satisfies $W_N(x) \to 1$ as x approaches the boundary of the safe set. In our analysis, we focus on the case (ii), which leads to the hyperbolic tangent transform.

Having established the construction of W_N, we now shift our focus to its approximation over the domain D, thereby laying the groundwork for characterizing the safe region in practical settings. In particular, we pose the following problem:

Problem 3.5.1. Given the control system in (2.6) and under the Zubov-type framework for safe regions (see (3.6) and (3.7)), the goal is to devise an algorithm that learns the transformed reciprocal barrier function $W_N(x)$ over D such that:

1. $W_N(x)$ is accurately approximated so that the safe set is characterized as in (3.6).
2. It can be used to construct a QP-based controller (2.28) that ensures system safety.

The proposed approach benefits from the advantages of the Zubov-type construction over traditional RBFs, which typically diverge at the boundary. Because W_N remains bounded and numerically more stable near the boundary ($\partial\mathcal{C}$), it is more convenient for practical learning methods.

Chapter 4

Methodology

4.1 Neural Network Training

We propose the following neural network training scheme to solve the above-described problem statement.

A physics-informed neural network (PINN) is employed to approximate the transformed reciprocal barrier function $W_N(x;\theta)$ that satisfies the Zubov-type PDE given in (3.7). We can write (3.7) as a general first-order PDE

$$F(x, W, \nabla W) = 0, \quad x \in D, \tag{4.1}$$

subject to the boundary condition

$$W_N(x;\theta) = \mathrm{bc}(x), \quad x \in \partial D. \tag{4.2}$$

We define $\mathrm{bc}(x)$ consistently with our construction as

$$\mathrm{bc}(x) = \begin{cases} 1, & \text{for } x \in \partial D \setminus \mathcal{C}, \\ \beta\Big(B(x)\Big), & \text{for } x \in \partial D \cap \mathcal{C}, \end{cases} \tag{4.3}$$

with β defined as in (3.8) and $B(x)$ being the reciprocal barrier function candidate.

To train $W_N(x;\theta)$ as an approximation of the solution to (4.1), we design a loss function comprising four components:

1. **PDE Residual Loss:** At a set of interior collocation points $\{x_i\}_{i=1}^{N_c} \subset D$, we

evaluate the residual using (4.1):

$$L_r = \frac{1}{N_c}\sum_{i=1}^{N_c}\Big[F\Big(x_i, W_N(x_i;\theta), \nabla W_N(x_i;\theta)\Big)\Big]^2. \tag{4.4}$$

2. **Boundary Loss:** At boundary points $\{y_i\}_{i=1}^{N_b} \subset \partial D$, we enforce the boundary condition by penalizing deviations from $\mathrm{bc}(y_i)$:

$$L_b = \frac{1}{N_b}\sum_{i=1}^{N_b}\Big(W_N(y_i;\theta) - \mathrm{bc}(y_i)\Big)^2. \tag{4.5}$$

3. **Safe Region Loss:** In a subset $S \subset D$ where additional safe region information is available (for example, near the equilibrium for an inverted pendulum), we impose that $W_N(x;\theta) \approx 0$. Thus,

$$L_{\text{safe}} = \frac{1}{|S|}\sum_{x_i \in S}\Big(W_N(x_i;\theta) - 0\Big)^2. \tag{4.6}$$

4. **Unsafe Region Loss:** In a subset $U \subset D$ where points are known to be unsafe (for example, near the boundary of $\mathcal{C}$), we enforce that $W_N(x;\theta) \approx 1$:

$$L_{\text{unsafe}} = \frac{1}{|U|}\sum_{x_i \in U}\Big(W_N(x_i;\theta) - 1\Big)^2. \tag{4.7}$$

Note that the specification of S and U is optional, but it helps the learning process if this information is made available. The overall loss function is then given by a weighted sum:

$$\mathcal{L}(\theta) = w_r\, L_r + w_b\, L_b + w_s\, L_{\text{safe}} + w_u\, L_{\text{unsafe}}, \tag{4.8}$$

where w_r, w_b, w_s, and w_u are positive weights chosen to balance the contributions of each term.

Training is performed according to Algorithm 1 using standard gradient descent methods (e.g., Adam optimizer) until convergence. An explicit implementation of this algorithm, in Python, for Unicycle system (see Section 5.2), is attached in the Appendix A.1.

4.2 Designing a Controller

We adopt the barrier-based control approach from Section 2.4.2, now employing our learned neural network $W_N(x;\theta)$ to define the barrier function. Recall from (2.17) that

Algorithm 1 PINN Training for Approximating $W_N(x;\theta)$ via the Zubov-Type PDE

Input: Interior collocation data $S_{\text{int}} = \{x_i\}_{i=1}^{N_c} \subset D$, Boundary data $S_{\text{b}} = \{(y_i, \text{bc}(y_i))\}_{i=1}^{N_b} \subset \partial D$, (Optional) Region-specific data for safe region $S_{\text{safe}} \subset D$ and unsafe region $S_{\text{unsafe}} \subset D$. Batch size B, learning rate η, maximum iterations M.
Output: Trained network $W_N(x;\theta^*)$ approximating the solution to (4.1).
Initialize network parameters θ.
Define loss function as in (4.8).
for $m = 1, 2, \ldots, M$ **do**
    Randomly sample batches: $S_m^{\text{int}} = \{x_i\}_{i=1}^{B}$ from S_{int}, S_m^{b} from S_{b}, S_m^{safe} from S_{safe}, and S_m^{unsafe} from S_{unsafe}.
    Compute $W_N(x_i;\theta)$ and its gradient $\nabla W_N(x_i;\theta)$ using automatic differentiation for each $x_i \in S_m^{\text{int}}$.
    Evaluate the PDE residual at each $x_i \in S_m^{\text{int}}$ using (4.1) and compute L_r as in (4.4).
    Evaluate the boundary loss L_b on the batch S_m^{b} using (4.5).
    (If applicable) Evaluate the safe and unsafe region losses L_{safe} and L_{unsafe} on S_m^{safe} and S_m^{unsafe} respectively.
    Compute the total loss $\mathcal{L}(\theta)$ for the current batch.
    Compute gradients $\nabla_\theta \mathcal{L}(\theta)$ via backpropagation.
    Update parameters: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta)$.
    Compute the current loss $\mathcal{L}_m = \mathcal{L}(\theta)$.
    if $\mathcal{L}_m$ is below a specified threshold **then**
        break
    end if
end for
Return: θ^* such that $W_N(x;\theta^*)$ approximates the solution of (4.1).

safety is enforced if the barrier condition holds. In our implementation, we first derive the barrier function $B(x;\theta)$ from the NN output via the transformation

$$B(x;\theta) = \frac{1}{2\alpha} \ln\left(\frac{1 + W_N(x;\theta)}{1 - W_N(x;\theta)}\right), \tag{4.9}$$

as given in (3.12) (written simply as $B(x)$ from now on). This transformation is chosen so that the level set $\{x \in D : W_N(x;\theta) < 1\}$ characterizes the safe set $\mathcal{C}$ in accordance with (3.6).

Using automatic differentiation, the gradient of B is computed as $\nabla B(x) = \frac{\partial B(x)}{\partial x}$. Whenever the safety condition (2.17) is violated (i.e., when

$$L_f B(x) + L_g B(x)\, u_{ref}(x) - \kappa\, h(x) > 0),$$

we apply a minimal-norm control adjustment Δu (as derived in (2.27)) to restore safety.

The resulting control law is then given by (2.28) and its implementation is detailed

in Algorithm 2. An explicit implementation of this algorithm, in Python, for Unicycle system (see Section 5.2), is attached in the Appendix A.2.

Algorithm 2 Barrier-Based Controller

1: **Input:** State x, nominal control $u_{ref}(x)$, learned NN output $W_N(x;\theta)$, parameters $\alpha, \kappa, \varepsilon$
2: **Output:** Corrected control $u(x)$
3: Compute $W \leftarrow W_N(x;\theta)$
4: Compute the barrier function according to (4.9).
5: Compute $\nabla B(x)$ via automatic differentiation.
6: Evaluate Lie derivatives of B according to (2.18).
7: Compute the safety measure:

$$s(x) = L_f B(x) + L_g B(x) u_{ref}(x) - \kappa\, h(x).$$

8: **If** $s(x) > 0$ **then** set

$$\Delta u = -\frac{s(x)}{\|L_g B(x)\|^2 + \varepsilon} L_g B(x)$$

9: **Else** set $\Delta u = 0$
10: Output $u(x) = u_{ref}(x) + \Delta u$

As in Section 2.4.2, the corrected control $u(x)$ modifies the nominal input $u_{ref}(x)$ only when necessary to preserve safety, thus ensuring that $\mathrm{Int}(\mathcal{C})$ remains forward invariant under the closed-loop dynamics.

4.3 Error Estimates and Convergence Analysis

In this section, we analyze the error and convergence properties of the neural network approximation $W_N(x;\theta)$ of the transformed reciprocal barrier function. Recall that the safe set is characterized by (3.6) and that $W_N(x)$ satisfies the Zubov-type PDE (3.7) with

$$\Psi(x) = \alpha\,(1 + W_N(x))\,\Phi(x),$$

as provided in (3.14). Moreover, the construction of $W_N(x)$ via the transformation (3.9) together with the choice $\psi(s) = \alpha(1+s)$ yields $\beta(s) = \tanh(\alpha s)$ (see (3.11) and (3.12)).

This choice implies that

$$G(s) = \big(1 - s\big)\,\psi(s) = \alpha\,(1 - s^2) \tag{4.10}$$

is monotonically decreasing on some interval $I \subseteq [0,1]$. For later reference, we adopt the following technical assumption:

Assumption. *There exists a nonempty interval* $I \subseteq [0,1]$ *such that the function* $G : I \to \mathbb{R}$ *defined by* $G(s) = \alpha\,(1 - s^2)$ *is monotonically decreasing.*

We now state the error estimate result for an ε-approximate solution of (3.7).

Proposition 4.3.1 (Error Estimate)**.** Let $\delta > 0$ be fixed such that the ball $\mathcal{B}_\delta$ is contained in $D \cap \mathrm{Int}(\mathcal{C})$. Suppose that $W_N(x)$ is the unique (viscosity) solution to (3.7) on D with boundary condition (4.2) and that $v(x)$ is an ε-approximate solution (in the sense of satisfying the PDE and boundary condition up to an error of ε and a boundary error of ε_b) on the set $D \setminus \overline{\mathcal{B}}_\delta$. Assume further that a condition analogous to the technical condition on G holds for $v(x)$. Then there exists a constant $C(\varepsilon, \varepsilon_b)$, with

$$C(\varepsilon, \varepsilon_b) \to 0 \quad \text{as} \quad \varepsilon \to 0 \quad \text{and} \quad \varepsilon_b \to 0,$$

such that

$$|W_N(x) - v(x)| \leq C(\varepsilon, \varepsilon_b), \quad \forall\, x \in \overline{D} \setminus \mathcal{B}_\delta. \tag{4.11}$$

Proof. A detailed proof follows similar lines as in the analysis of Zubov's equation in related work. Briefly, the argument relies on the monotonicity of $G(s)$, the continuity of $W_N(x)$, and standard comparison principles for viscosity solutions. (A complete proof is omitted here for brevity.) □

Next, we state the convergence result for a sequence of approximate solutions obtained through our neural network training scheme.

Theorem 4.3.2 (Convergence of Approximate Solutions)**.** Let $\{v_n\}$ be a sequence of ε_n-approximate solutions to (3.7) on D with a uniform Lipschitz constant, where $\varepsilon_n \to 0$ as $n \to \infty$. Suppose that the boundary conditions satisfy

$$v_n(x) \to W_N(x), \quad \text{uniformly for } x \in \partial D. \tag{4.12}$$

Then, for every $\delta > 0$ such that $\mathcal{B}_\delta \subset D \cap \mathrm{Int}(\mathcal{C})$, the sequence $\{v_n\}$ converges uniformly to $W_N(x)$ on $\overline{D}$, i.e.,

$$\lim_{n \to \infty} \sup_{x \in \overline{D}} |v_n(x) - W_N(x)| = 0. \tag{4.13}$$

Proof. The proof is based on a combination of the error estimate in Proposition 4.3.1 and standard arguments for the uniform convergence of viscosity solutions under a uniform Lipschitz condition and boundary consistency. For each fixed $\delta > 0$, one shows that the error can be made arbitrarily small on $\overline{D} \setminus \mathcal{B}_\delta$, and the local regularity of W_N in $D \cap \mathrm{Int}(\mathcal{C})$ ensures that the error near the interior of the safe set is also uniformly small. □

The results in Proposition 4.3.1 and Theorem 4.3.2 provide theoretical validation for the approximation method proposed in Section 4.1. They guarantee that the neural network solution $W_N(x;\theta)$ not only approximates the true solution of the Zubov-type PDE within a prescribed error margin but also converges uniformly under refinement of the approximation. This is crucial for ensuring the reliability of the barrier-based controller derived from the learned reciprocal barrier function.

Chapter 5

Experimental Results

In this section, we assess the efficacy of our proposed framework through three distinct case studies: an inverted pendulum system, autonomous ground navigation, and aerial navigation in obstacle-rich environments. All case studies are conducted on a computing platform equipped with an Intel i9-11900K CPU, 32GB RAM, and NVIDIA GeForce RTX 4090 GPU.

5.1 Inverted Pendulum

In this section, we present simulation results for an inverted pendulum example to demonstrate the proposed framework. The inverted pendulum dynamics are defined by

$$\dot{\theta} = \dot{\theta}, \quad \ddot{\theta} = \sin(\theta), \tag{5.1}$$

with state $x = [\theta,\, \dot{\theta}]^\top$. The safe region S is specified as

$$|\theta| \leq \frac{\pi}{8} \quad \text{and} \quad |\dot{\theta}| \leq 1, \tag{5.2}$$

while the unsafe region U is defined by

$$|\theta| \geq \frac{\pi}{2} \quad \text{or} \quad |\dot{\theta}| \geq 4. \tag{5.3}$$

A Barrier Network is trained using 10,000 random samples drawn uniformly from $\theta \in [-\pi, \pi]$ and $\dot{\theta} \in [-8, 8]$. The training follows the Algorithm (1) for 2000 epochs. A transformed barrier function is obtained via (4.9) which recovers an unbounded barrier function from the bounded network output $W_N(x)$. After training, $W_N(x)$ is evaluated on a grid over a restricted state space, and a contour plot is generated to visualize the

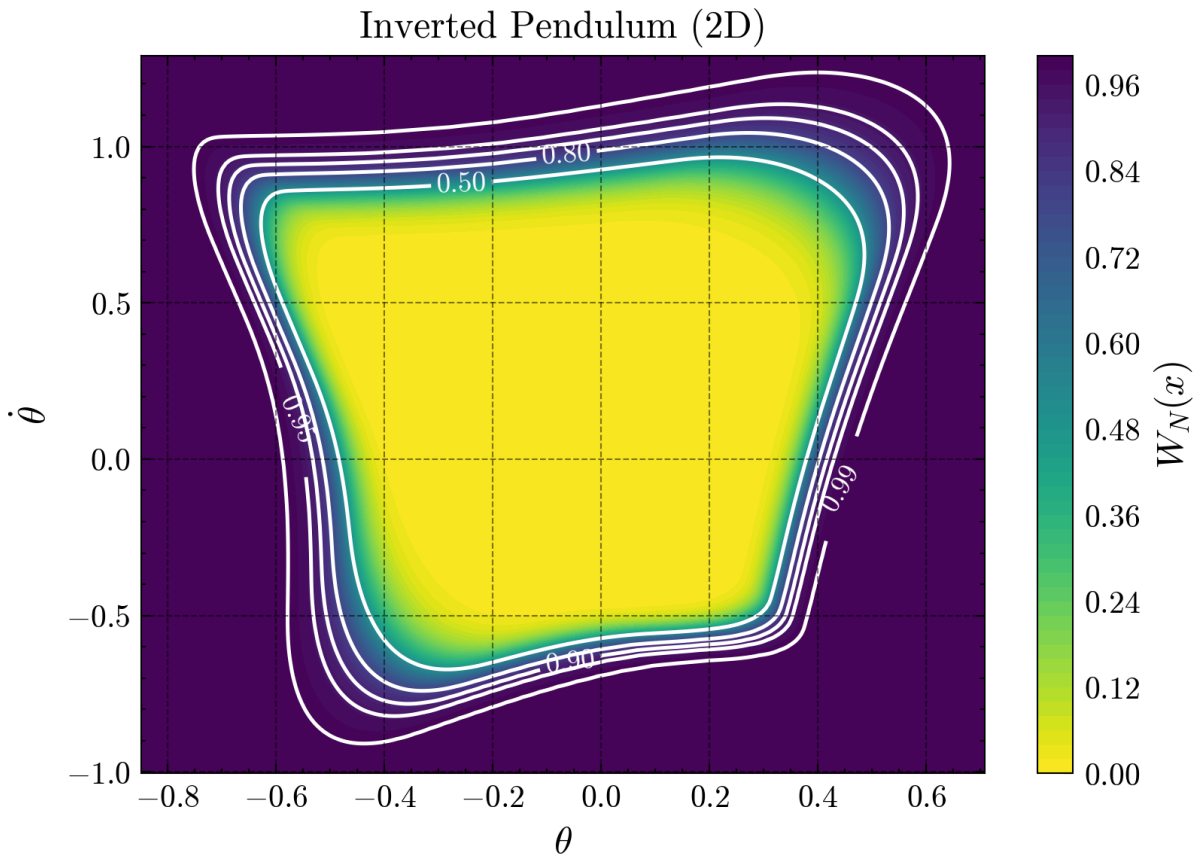


Figure 5.1: Contour plot of the learned barrier function $W_N(x)$ for inverted pendulum. Sub-level sets $\{x \in \mathbb{R}^n : W_N(x) \leq \gamma\}$ for increasing γ illustrate how the safe region expands, as we converge to a sub-level set.

learned barrier function.

As shown in Figure 5.1, $W_N(x)$ remains close to 0 in the safe region and approaches 1 near the unsafe boundaries. Sub-level sets $\{x \in \mathbb{R}^n : W_N(x) \leq \gamma\}$ for increasing γ illustrate how the area enclosed by safe region also increases.

From the simulation results, we also note another potential advantage of Zubov's construction of a Barrier function, namely $W_N(x)$, that it is bounded and its value approaches one as x approaches the boundary, whereas the traditional Reciprocal Barrier Functions (RBFs) approach infinity as x tends to the boundary. Having a bounded function offers advantages in numerical approximation and makes it possible to extend its domain to the entire state space. By varying the threshold on $W_N(x)$, we can also adjust the effective safe region, thereby giving us flexibility in terms of choosing less (or more) restrictive safety specifications.

5.2 Autonomous Ground Navigation

5.2.1 Simulation Results

In this section, we present simulation results for an autonomous ground navigation system using the learned neural network reciprocal barrier function (NN RBF) to enforce safety. A representative setup is shown in Figure 5.2 [11, 41, 42]:

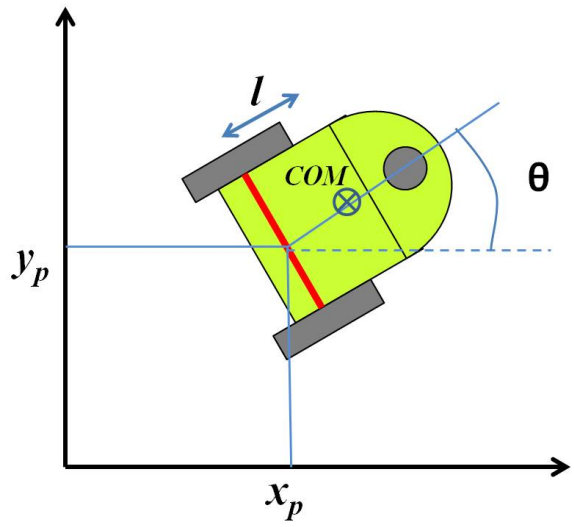


Figure 5.2: Representative ground navigation setup. This figure is for illustration purposes only, please refer to the text for the correct notation and mathematical details.

Our autonomous ground navigation robot is governed by the control-affine system

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} v\cos(\psi) \\ v\sin(\psi) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u, \tag{5.4}$$

where the control influence only enters the orientation dynamics. Here, $v > 0$ is the constant forward speed, and the control input u is a scalar adjusting the robot's heading. The state $x \in \mathbb{R}^3$, where $x = [x_1, x_2, \psi]^\top$, is sampled uniformly from

$$x_1 \in [-2, 2], \quad x_2 \in [-2, 2], \quad \psi \in [-\pi, \pi]. \tag{5.5}$$

The safe region S is defined by

$$|x_1| \geq 1.5 \text{ or } |x_2| \geq 1.5, \tag{5.6}$$

while the unsafe region U is characterized as

$$|x_1| \leq 0.2 \text{ and } |x_2| \leq 0.2. \tag{5.7}$$

A Barrier Network is trained using 10,000 samples drawn uniformly from the above state space.

A simple reference controller is designed to steer the robot toward a desired goal (here taken as $(1, 1)$). The reference controller computes a nominal input u_{ref} based on the angular error:

$$u_{\text{ref}} = K \left(\arctan\left(\frac{1 - x_2}{1 - x_1} \right) - \psi \right), \tag{5.8}$$

with gain $K > 0$.

To enforce safety, a learned NN RBF $W_N(x; \theta)$ is obtained using the PINN training algorithm described in Algorithm 1 for 2000 epochs. The barrier-based controller (Algorithm 2) then computes a correction Δu by enforcing the safety condition. In this way, the barrier-based controller adjusts the nominal control to maintain safety, i.e., to ensure that the system avoids an unsafe region (here specified as a rectangular area, e.g., $[–0.2, 0.2] \times [-0.2, 0.2]$).

Figure 5.3 shows that we converge to the expected CBF shape for an autonomous ground navigation robot as $W_N(x) \to 1$. Sub-level sets $\{x \in \mathbb{R}^n : W_N(x) \leq \gamma\}$ for increasing γ illustrate how the area enclosed by the contours, indicating the unsafe region in this case, decreases, showing that we are able to recover a larger safe region. This also gives us flexibility in choosing a less or more conservative barrier function by selecting the appropriate γ value for the underlying sub-level set.

Figures 5.4 and 5.5 illustrate the marked contrast between the baseline approach and our proposed method. As shown in Fig. 5.4, although the baseline ensures that the zero-level set does not overlap with the unsafe region, it does not fully enclose it, thereby yielding only a limited safe subset of the state space. In contrast, Fig. 5.5 demonstrates how our learned barrier encapsulates the unsafe region entirely in a cylindrical level set, leading to a much larger effective safe region. This highlights the flexibility of our proposed approach, which is not constrained by a prescribed functional form and can effectively encode the desired safety constraints. For reference, we employ the same baseline as in [43, 23].

Figure 5.6 shows the simulated trajectories of the autonomous ground navigation robot. The blue trajectory corresponds to the reference controller alone, while the magenta trajectory shows the result when the barrier-based correction is applied. The unsafe region is highlighted by the red rectangle, and the start and goal positions are indicated by

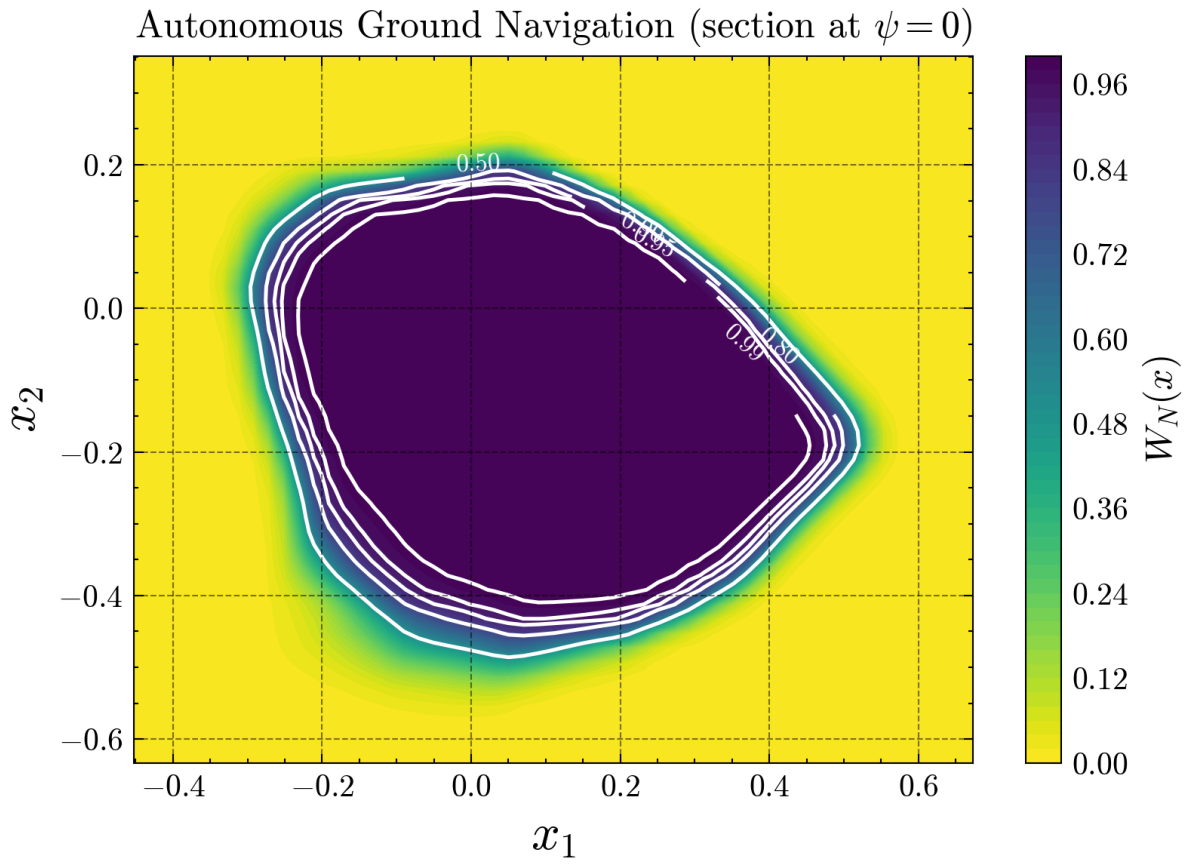


Figure 5.3: Contour plot of $W_N(x)$ at $\psi = 0$ for the autonomous ground navigation system. Sub-level sets $\{x \in \mathbb{R}^n : W_N(x) \leq \gamma\}$ for increasing γ illustrate how the unsafe region shrinks (leading to a larger safe area outside the contour), as we converge to a sub-level set.

green and red markers, respectively. We can see that with the barrier-adjusted controller, the robot is able to avoid the obstacle/unsafe region cleanly.

5.2.2 Hardware Results

The experiments were performed on Unitree Go1 [44], a Quadrupedal Robot, to demonstrate the efficacy of our Zubov NCBF on Unicycle models. In these experiments, the robot's global position, as well as the position of gray obstacle present, was accurately measured using the PhaseSpace™ motion capture system. This system operates at tracking frequency of 960 Hz.

To obtain real-time state estimates, two LED markers were affixed to the front and rear of the robot. These markers provided reliable measurements of the robot's key state variables, including position, translational velocity, orientation, and angular velocity. Meanwhile, the onboard control loop was executed at 100 Hz, enabling corrections during obstacle avoidance maneuvers.

Hardware trials confirmed that the barrier-based controller adjusted the nominal con-

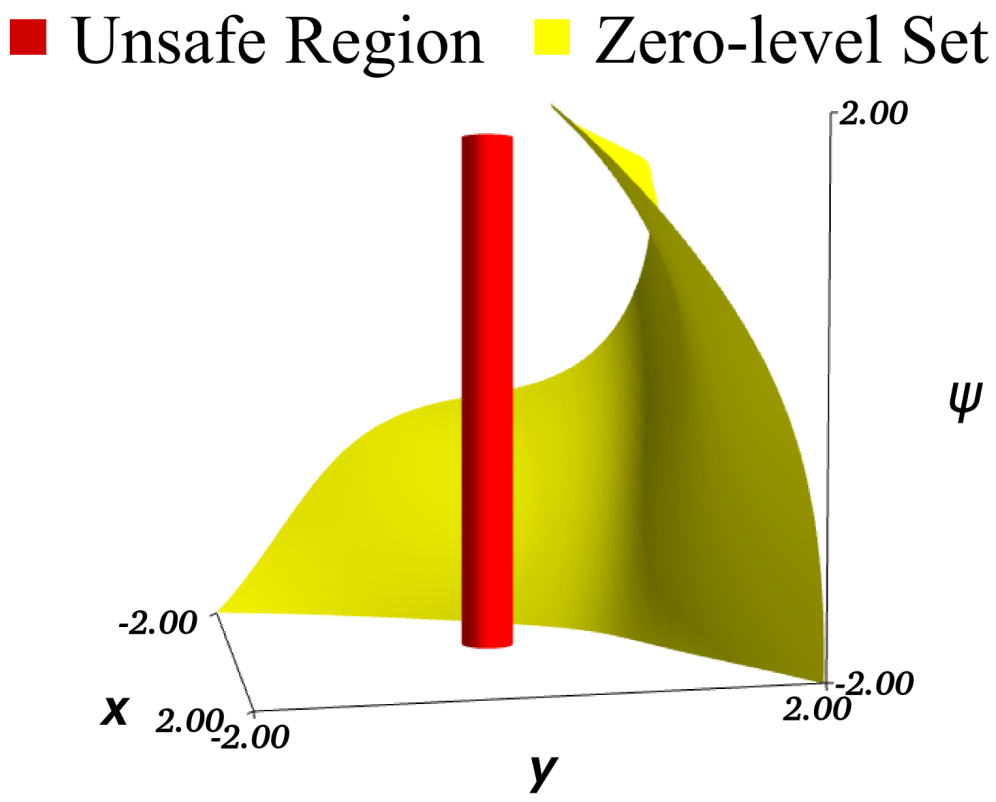


Figure 5.4: Zero-level set obtained using the baseline approach [43]. Although the set does not intersect the unsafe region, it fails to enclose it completely, resulting in only a small portion of the state space being certified as safe.

trol inputs in real time, ensuring that the robot navigated safely around obstacles. The experimental outcomes on the Unitree Go1 (Figure 5.7) are consistent with our simulation results (Figure 5.6), thereby validating the practical viability of the Zubov-NCBF approach for autonomous ground navigation.

5.3 Aerial navigation in obstacle-rich environments

In this section, we present simulation results for an aerial navigation system using a learned neural network reciprocal barrier function (NN RBF) to enhance safety.

The aerial navigation vehicle is modeled as a six-dimensional system with state

$$x = \begin{bmatrix} y & z & \phi & v_y & v_z & \dot{\phi} \end{bmatrix}^\top \in \mathbb{R}^6, \tag{5.9}$$

where (y, z) denote the planar positions, ϕ is the pitch angle, (v_y, v_z) are the linear velocities, and $\dot{\phi}$ is the angular velocity.

A representative setup is shown in Figure 5.8 [45]:

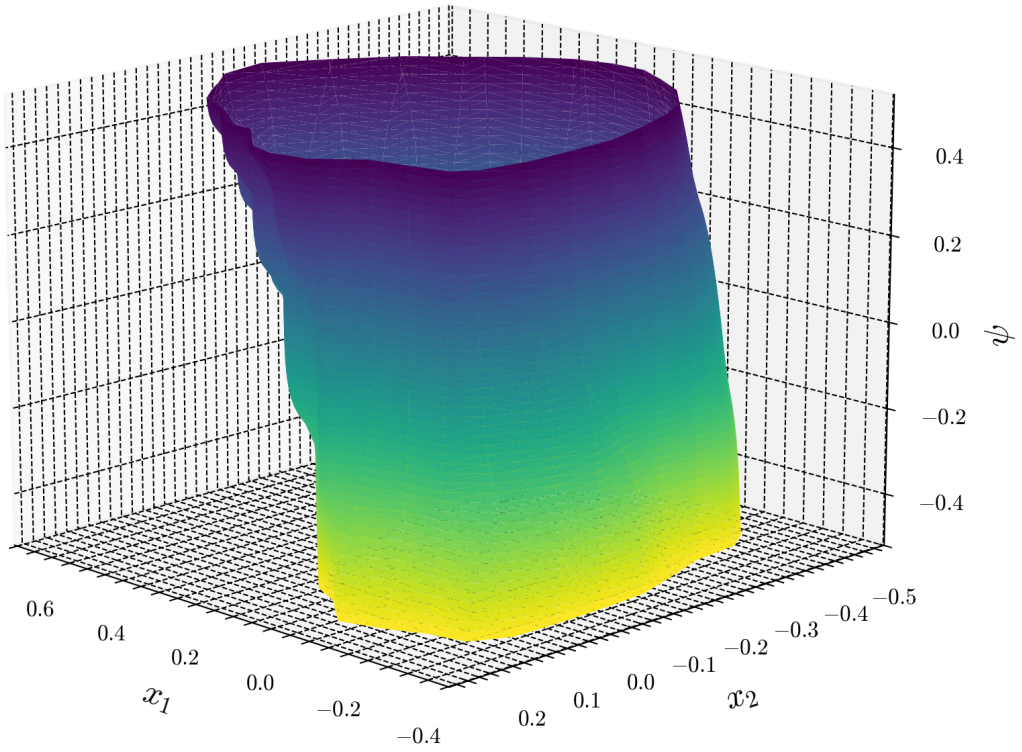


Figure 5.5: A 3D visualization of the level set $\{x : W_N(x) = 0.5\}$ for the unicycle dynamics. The unsafe region is fully contained within the cylindrical structure of the barrier, thus recovering a significantly larger safe region compared to the baseline.

The system dynamics are given by

$$\begin{bmatrix} \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{v}_y \\ \dot{v}_z \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} v_y \\ v_z \\ \dot{\phi} \\ 0 \\ -g \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -\sin\phi & -\sin\phi \\ \cos\phi & \cos\phi \\ 1 & -1 \end{bmatrix} u. \tag{5.10}$$

Here, $g = 9.81$ is the gravitational constant and the control input $u \in \mathbb{R}^2$ corresponds to rotor speeds that are indirectly related to forces and torques acting on the vehicle.

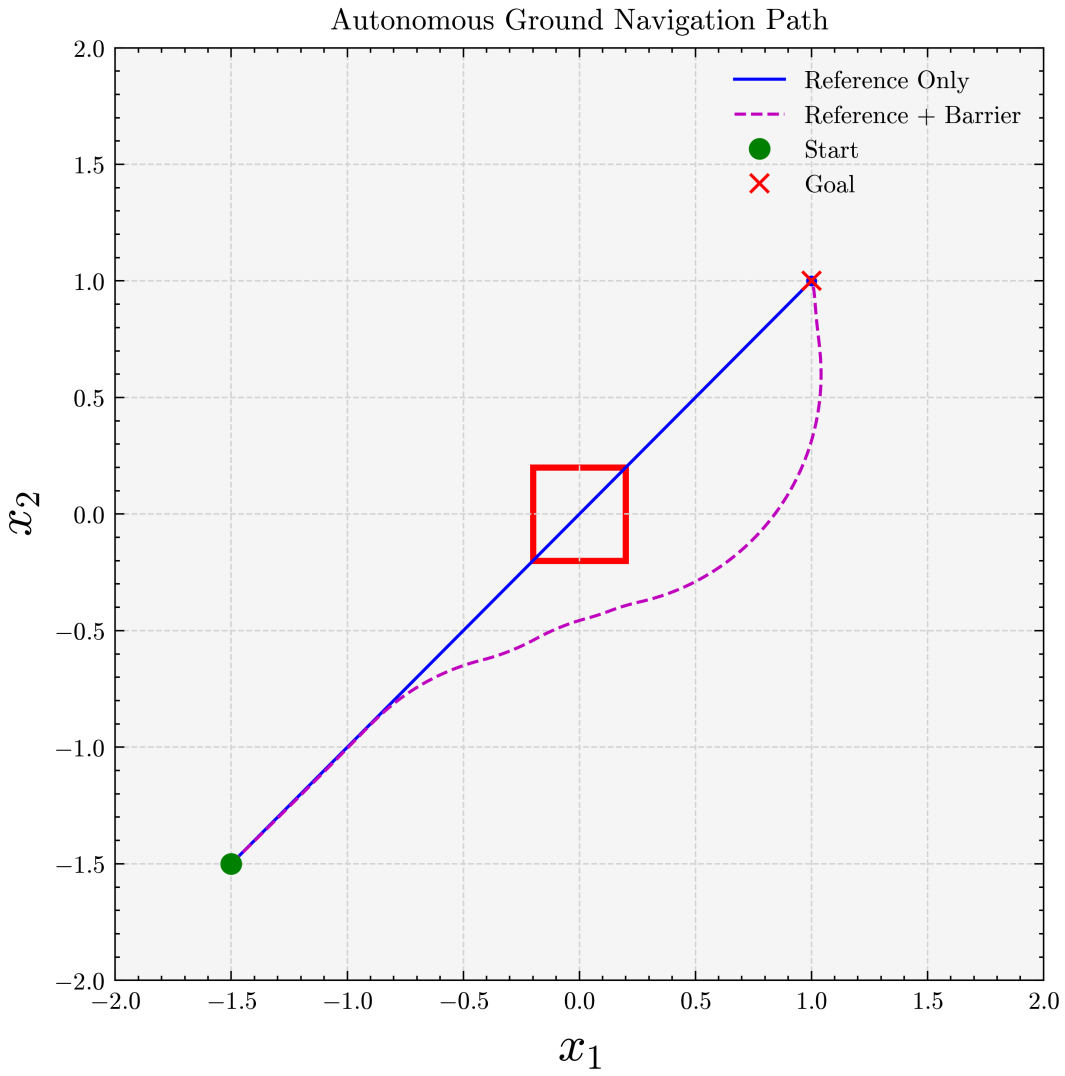


Figure 5.6: Simulated autonomous ground navigation robot trajectories demonstrating obstacle avoidance when a barrier-based controller is employed.

5.3.1 Reference Controller Design

The reference controller is designed using a PD+I architecture, which computes the nominal control input by actively reducing the error between the current state and the desired target. In our implementation, the controller is tasked with steering the vehicle from an initial state $x_0 = \left(y_0,\ z_0,\ \phi_0,\ v_{y,0},\ v_{z,0},\ \dot{\phi}_0\right)^\top$ to a desired target state given by $x_d = \left(y_d,\ z_d,\ 0,\ 0,\ 0,\ 0\right)^\top$. Here, y and z denote the planar positions, ϕ is the pitch angle, while v_y, v_z and $\dot{\phi}$ represent the linear and angular velocities, respectively.

At each control cycle, the controller evaluates the errors in position and orientation. For the planar positions, the errors are computed as $e_y = y_d - y$ and $e_z = z_d - z$. These errors are then processed by the proportional, derivative, and integral terms, yielding the



Figure 5.7: Autonomous ground navigation in obstacle laden environment. Continuous video frames (left to right).

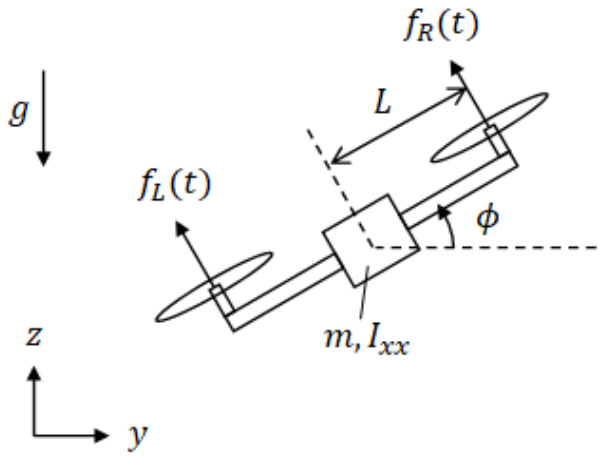


Figure 5.8: Representative aerial navigation setup. This figure is for illustration purposes only, please refer to the text for the correct notation and mathematical details.

desired accelerations:

$$a_{y,\text{des}} = a_{y,\text{target}} + p_y\, e_y + d_y\, (v_{y,\text{target}} - v_y) + i_y \int e_y\, dt, \tag{5.11}$$

$$a_{z,\text{des}} = a_{z,\text{target}} + p_z\, e_z + d_z\, (v_{z,\text{target}} - v_z) + i_z \int e_z\, dt. \tag{5.12}$$

The gains p_y, d_y, i_y and p_z, d_z, i_z are tuned to achieve a balance between rapid response and smooth convergence, while the integral action serves to mitigate any steady-state error that might arise from model uncertainties or external disturbances.

An important feature of the controller is the transformation of the desired lateral acceleration $a_{y,\text{des}}$ into a corresponding desired pitch angle. This is achieved using the relation $\phi_{\text{des}} = -\frac{a_{y,\text{des}}}{g + a_{z,\text{des}} + \epsilon}$, where $g = 9.81\,\text{m/s}^2$ is the gravitational constant and ϵ is a small number ensuring numerical stability. The difference between the desired pitch ϕ_{des} and the actual pitch ϕ is then used to compute a corrective acceleration for rotational

dynamics by applying a similar PD+I formulation:

$$a_{\phi,\text{des}} = p_\phi \left(\phi_{\text{des}} - \phi\right) + d_\phi \frac{d}{dt}(\phi_{\text{des}} - \phi) + i_\phi \int (\phi_{\text{des}} - \phi)\, dt. \tag{5.13}$$

With the desired accelerations in hand, the total thrust required by the vehicle is calculated as $T = m\sqrt{a_{y,\text{des}}^2 + (g + a_{z,\text{des}})^2}$, where m is the mass of the vehicle. The torque necessary for achieving the desired rotational acceleration is computed by $\tau = I\, a_{\phi,\text{des}}$, with I being the moment of inertia. These computed thrust and torque values are then mapped to individual rotor forces using the following expressions:

$$F_1 = \frac{1}{2}\left(T - \frac{\tau}{L}\right), \quad F_2 = \frac{1}{2}\left(T + \frac{\tau}{L}\right), \tag{5.14}$$

where L denotes the effective arm length of the vehicle.

Finally, the rotor forces are converted into rotor speeds by the relations $\omega_i = \sqrt{F_i/k_f}, \quad i = 1, 2$, where k_f is a constant relating the force produced by the rotors to their rotational speed. The controller continuously updates the integrator states based on the accumulated errors at each time step Δt, thus refining the control inputs over time.

5.3.2 Barrier-Based Controller Integration

Safety is enforced by augmenting the nominal control with a barrier-based correction, as described in Algorithm 2. The learned NN RBF (obtained via Algorithm 1) is used to compute the barrier function $B(x)$ and its gradient.

5.3.3 Simulation Setup and Results

Simulations are performed using the system dynamics described above. The initial state is set to

$$x_0 = [-1.5,\ -1.5,\ 0,\ 0,\ 0,\ 0]^\top,$$

and the simulation is run for $T_{\text{final}} = 10\,\text{s}$ with a time step of $\Delta t = 0.01\,\text{s}$. Two scenarios are compared:

- **Reference-Only Control:** The vehicle is controlled solely by the PID controller.
- **Reference + Barrier Control:** The barrier-based controller augments the nominal controller to enforce safety.

Figure 5.9 shows that W_N converges to a well-behaved barrier shape as $W_N(x) \to 1$. Sub-level sets $\{x \in \mathbb{R}^n : W_N(x) \leq \gamma\}$ for increasing γ illustrate how the area enclosed by

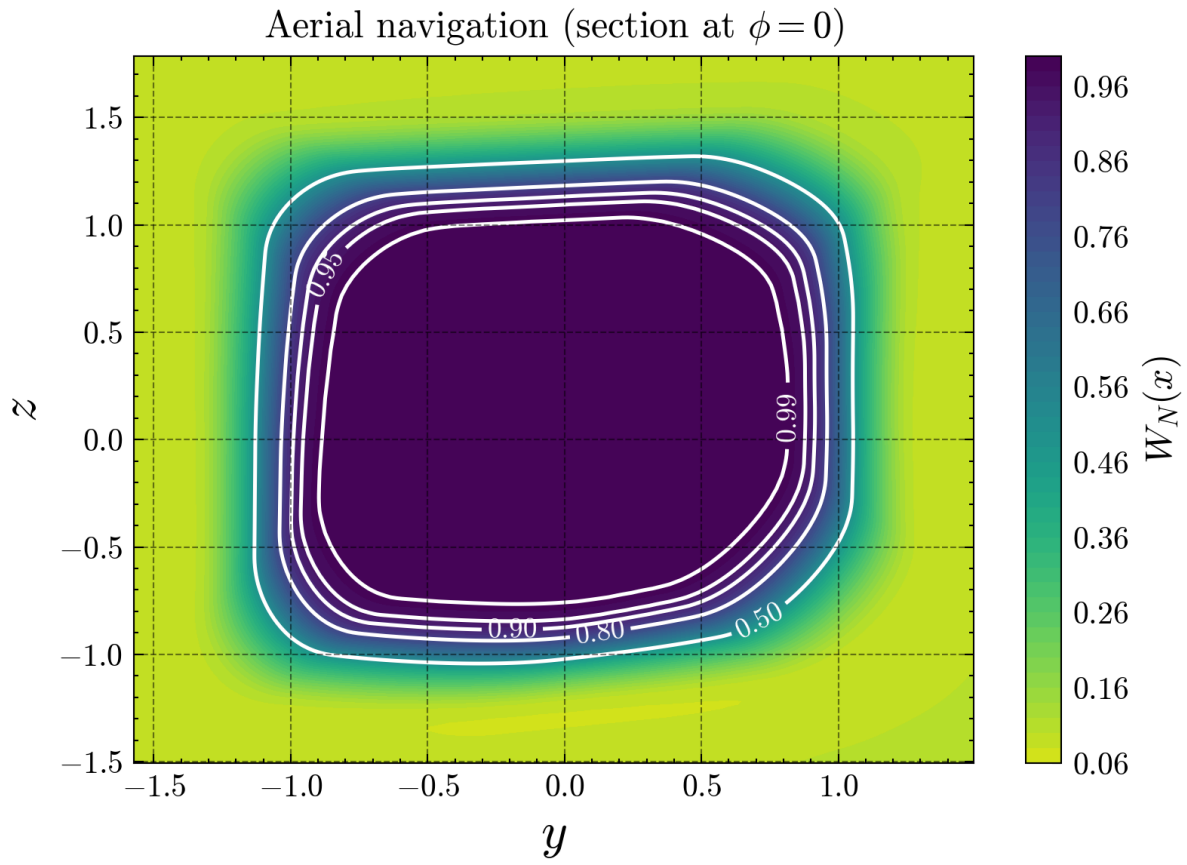


Figure 5.9: Contour plot of $W_N(x)$ at $\phi = 0$ for the aerial navigation vehicle (other dimensions not shown). Sub-level sets $\{x \in \mathbb{R}^n : W_N(x) \leq \gamma\}$ for increasing γ illustrate how the unsafe region shrinks (leading to a larger safe area outside the contour), as we converge to a sub-level set.

the contours, indicating the unsafe region, decreases, showing that we are able to recover a larger safe region. This flexibility enables us to assign an underlying γ value of our choosing based on the safety requirements of the deployment at hand.

Figure 5.10 depicts the three-dimensional shape of the level set $\{x : W_N(x) = 0.5\}$, spanning the y, z, and ϕ dimensions. Similar to Figure 5.5, the level set fully encloses the unsafe region, thereby yielding a substantially larger safe region in the state space. Moreover, the smooth curvature of the learned function W_N is readily observed, leading to well-behaved gradients that facilitate real-time controller corrections and promote stable, safety-preserving maneuvers during flight.

Figure 5.11 shows the resulting trajectories in the y-z plane. The unsafe region, defined as the rectangle, is depicted in red. The reference-only trajectory is shown in blue, while the trajectory with the barrier correction is shown in magenta. The start and target positions are marked with green and red symbols, respectively. We can clearly see that the barrier-adjusted controller effectively avoids the obstacle/unsafe region while maintaining a reasonable trajectory that reaches the goal. The result of the aerial navigation system

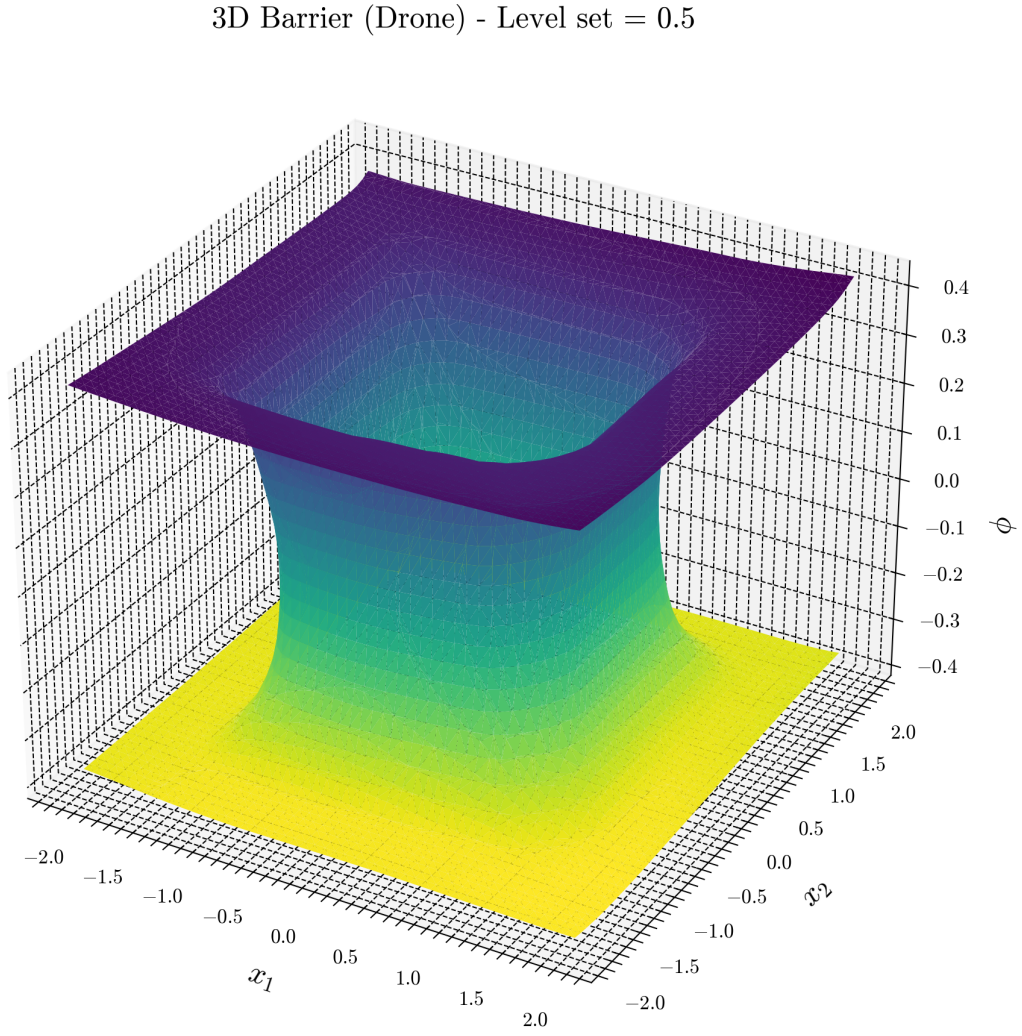


Figure 5.10: A 3D visualization of the level set $\{x : W_N(x) = 0.5\}$ for the aerial navigation vehicle, with axes corresponding to y, z, and ϕ.

is especially remarkable because it demonstrates efficient learning of a Control Barrier Function in a six-dimensional state space, a task many modern learning-based approaches struggle with [12, 13].

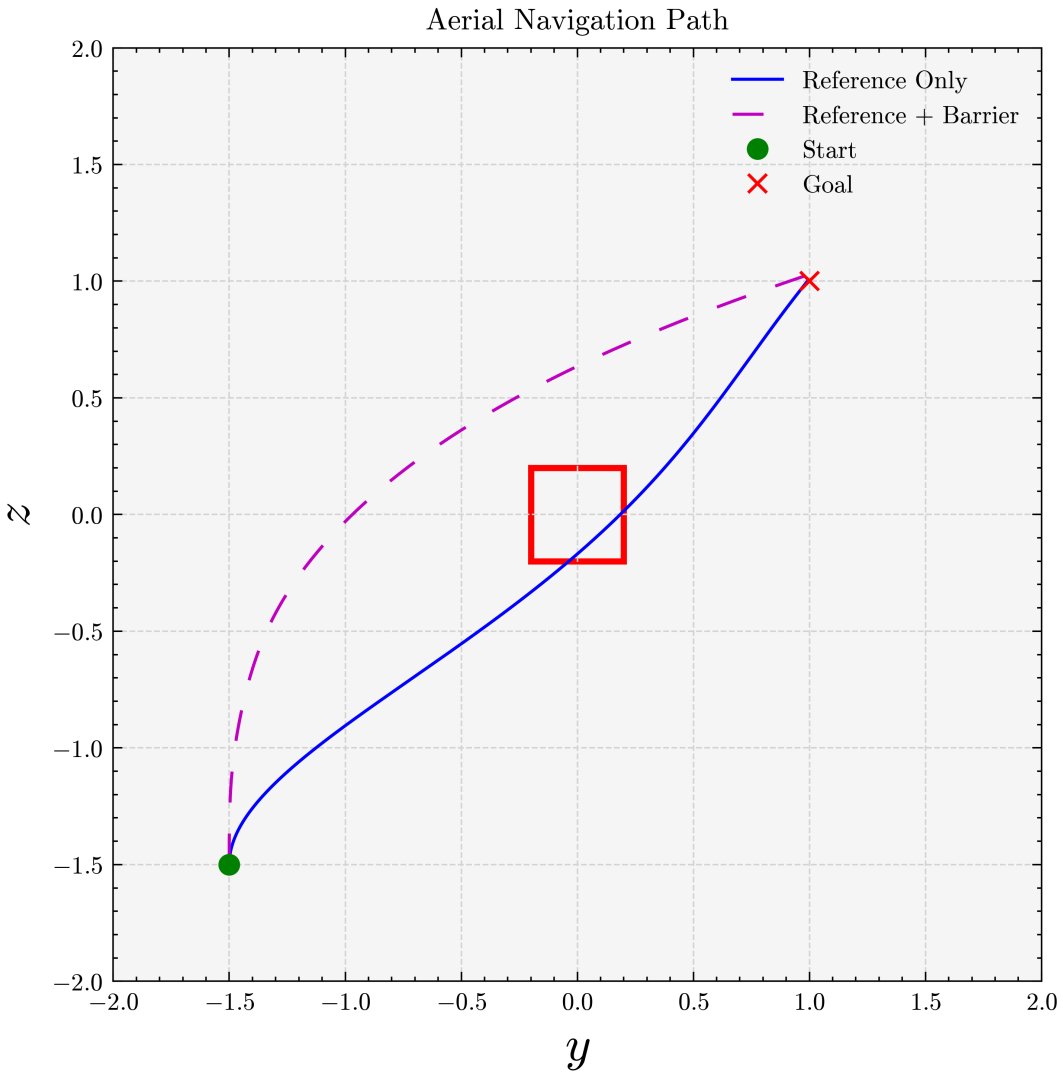


Figure 5.11: Trajectories of the aerial navigation vehicle in the y-z plane, using a trajectory following reference controller. Successful avoidance of the unsafe region is observed in the barrier-based controller due to safety adjustments to the control inputs.

Chapter 6

Conclusion and Future Work

6.1 Conclusions

In this work, we presented a systematic approach to reformulate Zubov's Partial Differential Equation (PDE) within the context of safety and leveraged this formulation to synthesize neural Control Barrier Functions (CBFs) using physics-informed neural networks. Furthermore, we utilized the learned neural CBFs to design a safe controller through a Quadratic Program (QP). A key advantage of our framework lies in its flexibility, allowing the definition of custom level sets that enable users to adapt the safe region according to specific safety requirements. The developed algorithms demonstrate efficient training, exhibiting rapid convergence and high sample efficiency, with only 10,000 random samples and 2,000 training epochs across diverse environments. To evaluate the efficacy of our approach, we conducted experiments on various control tasks, including an inverted pendulum, autonomous ground navigation, and aerial navigation in obstacle-laden environments. For the ground navigation task, the controller was also deployed on a Unitree Go1 robot. The results of aerial navigation task demonstrate that our method effectively scales to high-dimensional systems while ensuring safety.

6.2 Future work

Future work may explore:

- Modifying the loss function under the proposed training scheme to account for multiple constraints by learning a single W_N function instead of composing multiple barrier functions [46, 47].
- Extending our approach from Reciprocal Barrier Functions (RBFs) to Zeroing Barrier Functions (ZBFs) [7, 10].

- Extensions to more complex systems and incorporating robustness against model uncertainties and verification schemes to provide formal guarantees on the learned Neural CBF.

Appendix A

Code Implementations

A.1 Algorithm 1: Unicycle Implementation

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

class BarrierNetwork(nn.Module):
    def __init__(self):
        super(BarrierNetwork, self).__init__()
        self.fc1 = nn.Linear(3, 64)
        self.fc2 = nn.Linear(64, 128)
        # self.fc3 = nn.Linear(128, 128)
        self.fc4 = nn.Linear(128, 64)
        self.fc5 = nn.Linear(64, 1)
        self.elu = nn.ELU()

    def forward(self, x):
        x = self.elu(self.fc1(x))
        x = self.elu(self.fc2(x))
        # x = self.elu(self.fc3(x))
        x = self.elu(self.fc4(x))
        x = torch.sigmoid(self.fc5(x))
        return x

alpha = 1
kappa = 10
```

```
v = 1.0  # constant forward speed for the unicycle

x1_min, x1_max = -2, 2
x2_min, x2_max = -2, 2
psi_min, psi_max = -3.14, 3.14

num_samples = 10000
batch_size = 256
epochs = 2000
learning_rate = 1e-3

x1_vals = np.random.uniform(x1_min, x1_max, num_samples)
x2_vals = np.random.uniform(x2_min, x2_max, num_samples)
psi_vals = np.random.uniform(psi_min, psi_max, num_samples)
samples = np.stack([x1_vals, x2_vals, psi_vals], axis=1)
X = torch.tensor(samples, dtype=torch.float32, device=device)

def is_in_safe_region(points):
    x1, x2 = points[:, 0], points[:, 1]
    return (x1 < -1.5) | (x1 > 1.5) | (x2 < -1.5) | (x2 > 1.5)

def is_in_unsafe_region(points):
    x1, x2 = points[:, 0], points[:, 1]
    return (x1 >= -0.2) & (x1 <= 0.2) & (x2 >= -0.2) & (x2 <= 0.2)

safe_mask = is_in_safe_region(X).bool()
unsafe_mask = is_in_unsafe_region(X).bool()

def f_drift(x):
    # dot_x = v*cos(psi), dot_y = v*sin(psi), dot_psi = 0.
    psi = x[:, 2]
    dot_x = v * torch.cos(psi)
    dot_y = v * torch.sin(psi)
    dot_psi = torch.zeros_like(psi)
    return torch.stack([dot_x, dot_y, dot_psi], dim=1)

def g_drift(x):
    # g(x) = [0, 0, 1] (with shape (batch, 3, 1)).
    batch_size = x.shape[0]
    g_out = torch.zeros(batch_size, 3, 1, device=x.device)
    g_out[:, 2, 0] = 1.0
    return g_out

def compute_B_from_W(W):
    W_clamped = torch.clamp(W, -0.9999, 0.9999)
    B = (1/alpha)*0.5 * torch.log((1 + W_clamped) / (1 - W_clamped))
    return B
```

```
77
78  def compute_dBdx_from_gradW(W, grad_W):
79      W_clamped = torch.clamp(W, -0.9999, 0.9999)
80      denom = (1 - W_clamped**2).unsqueeze(-1) + 1e-12
81      dBdW = 1.0 / (alpha * denom)
82      dBdx = grad_W * dBdW
83      return dBdx
84
85  def compute_loss_and_control(network, X_batch, safe_m, unsafe_m,
86                               w_r, w_b, w_d, kappa):
87      X_batch.requires_grad_(True)
88      W = network(X_batch).squeeze()
89      grad_W = torch.autograd.grad(
90          outputs=W,
91          inputs=X_batch,
92          grad_outputs=torch.ones_like(W),
93          create_graph=True
94      )[0]
95      B = compute_B_from_W(W)
96      dBdx = compute_dBdx_from_gradW(W, grad_W)
97
98      f_val = f_drift(X_batch)
99      g_val = g_drift(X_batch)  # shape: (batch, 3, 1)
100
101     u_ref = torch.zeros(X_batch.shape[0], 1, device=X_batch.device).
            uniform_(0, 10)
102     LfB = torch.sum(dBdx * f_val, dim=1)
103     LgB = torch.einsum('bi,bij->bj', dBdx, g_val)  # shape: (batch, 1)
104
105     gamma = (LfB + torch.sum(LgB * u_ref, dim=1) - kappa * B)
106     LgB_norm_sq = torch.sum(LgB**2, dim=1) + 1e-6
107     u_values = (gamma / LgB_norm_sq).unsqueeze(-1) * LgB  # shape: (
            batch, 1)
108
109     u_expanded = u_values.unsqueeze(-1)  # shape: (batch, 1, 1)
110     g_u = torch.bmm(g_val, u_expanded).squeeze(-1)  # shape: (batch, 3)
111     fX_batch = f_val + g_u
112     residual = torch.sum(grad_W * fX_batch, dim=1)
113     L_r = torch.mean(residual**2)
114     if torch.any(unsafe_m):
115         L_b = torch.mean((W[unsafe_m] - 1)**2)
116     else:
117         L_b = torch.tensor(0.0, device=X_batch.device)
118     if torch.any(safe_m):
119         L_d = torch.mean((W[safe_m] - 0)**2) #changed
120     else:
121         L_d = torch.tensor(0.0, device=X_batch.device)
```

```
122     loss = w_r * L_r + w_b * L_b + w_d * L_d
123     return loss, L_r.detach(), L_b.detach(), L_d.detach()
124 
125 network = BarrierNetwork().to(device)
126 optimizer = optim.Adam(network.parameters(), lr=learning_rate)
127 for p in network.parameters():
128     nn.init.normal_(p, 0, 0.1)
129 
130 safe_mask = safe_mask.to(device)
131 unsafe_mask = unsafe_mask.to(device)
132 indices_all = np.arange(num_samples)
133 
134 for epoch in range(epochs):
135     np.random.shuffle(indices_all)
136     epoch_loss = 0.0
137     pde_loss = 0.0
138     safe_loss = 0.0
139     unsafe_loss = 0.0
140     for i in range(0, num_samples, batch_size):
141         batch_indices = indices_all[i:i+batch_size]
142         X_batch = X[batch_indices]
143         batch_safe = safe_mask[batch_indices]
144         batch_unsafe = unsafe_mask[batch_indices]
145         # warm start.
146         if epoch < 50:
147             w_r, w_b, w_d = 0.0, 1.0, 1.0
148         else:
149             w_r, w_b, w_d = 0.1, 1.0, 5.0 #changed
150         loss, lr_, lb_, ld_ = compute_loss_and_control(
151             network, X_batch, batch_safe, batch_unsafe,
152             w_r, w_b, w_d, kappa
153         )
154         optimizer.zero_grad()
155         loss.backward()
156         optimizer.step()
157         epoch_loss += loss.item()
158         pde_loss += lr_
159         safe_loss += ld_
160         unsafe_loss += lb_
161     if epoch % 10 == 0:
162         print(f"Epoch␣{epoch}/{epochs},␣Loss:␣{epoch_loss:.4f},␣PDE␣
            Loss:␣{pde_loss:.4f},␣SAFE␣Loss:␣{safe_loss:.4f},␣UNSAFE␣
            Loss:␣{unsafe_loss:.4f}")
163 
164 dtime = datetime.now()
165 MODEL_PATH = f"barrier_network_unicycle_final_{dtime}.pt"
166 torch.save(network.state_dict(), MODEL_PATH)
```

Listing A.1: Algorithm 1 Implementation for Unicycle

A.2 Algorithm 2: Unicycle Implementation

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
import math
from datetime import datetime

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

class BarrierNetwork(nn.Module):
    def __init__(self):
        super(BarrierNetwork, self).__init__()
        self.fc1 = nn.Linear(3, 64)
        self.fc2 = nn.Linear(64, 128)
        # self.fc3 = nn.Linear(128, 128)
        self.fc4 = nn.Linear(128, 64)
        self.fc5 = nn.Linear(64, 1)
        self.elu = nn.ELU()

    def forward(self, x):
        x = self.elu(self.fc1(x))
        x = self.elu(self.fc2(x))
        # x = self.elu(self.fc3(x))
        x = self.elu(self.fc4(x))
        x = torch.sigmoid(self.fc5(x))
        return x

def compute_B_and_derivatives(X, network, alpha=1.0):
    W = network(X).squeeze(-1)
    W = torch.clamp(W, -0.9999, 0.9999)
    B = (1/(2*alpha)) * torch.log((1+W)/(1-W))
    grad_W = torch.autograd.grad(outputs=W, inputs=X, grad_outputs=
        torch.ones_like(W), create_graph=True)[0]
    dBdx = grad_W / (alpha * (1 - W**2))
    return B, dBdx

def unicycle_f(x, v=1.0):
    psi = x[:, 2]
    dot_x = v * torch.cos(psi)
```

```
    dot_y = v * torch.sin(psi)
    dot_psi = torch.zeros_like(psi)
    return torch.stack([dot_x, dot_y, dot_psi], dim=1)

def unicycle_g(x):
    batch = x.shape[0]
    g_out = torch.zeros(batch, 3, 1, device=x.device, dtype=x.dtype)
    g_out[:, 2, 0] = 1.0
    return g_out

def simple_ref_controller(x, K=66.0):
    psi = x[:, 2]
    dx = 1.0 - x[:, 0]
    dy = 1.0 - x[:, 1]
    psi_des = torch.atan2(dy, dx)
    e = psi_des - psi
    e = (e + math.pi) % (2 * math.pi) - math.pi
    return K * e

def unicycle_barr_controller(X, network, u_ref=None, alpha=1.0, kappa
    =10.0):
    batch = X.shape[0]
    if u_ref is None:
        u_ref = torch.zeros(batch, 1, dtype=X.dtype, device=X.device)
    else:
        if not torch.is_tensor(u_ref):
            u_ref = torch.tensor(u_ref, dtype=X.dtype, device=X.device)
                .unsqueeze(0).repeat(batch, 1)
        elif u_ref.dim() == 1:
            u_ref = u_ref.unsqueeze(1)
    X = X.clone().detach().requires_grad_(True)
    B, dBdx = compute_B_and_derivatives(X, network, alpha=alpha)
    f_val = unicycle_f(X)
    g_val = unicycle_g(X)
    LfB = torch.sum(dBdx * f_val, dim=1)
    LgB = torch.bmm(dBdx.unsqueeze(1), g_val)
    safety_val = LfB + LgB.squeeze(1) * u_ref.squeeze(1) - kappa * B
    eps = 1e-6
    LgB_norm_sq = (LgB**2).squeeze(1) + eps
    correction = - (safety_val / LgB_norm_sq).unsqueeze(1) * LgB
    correction = torch.where(safety_val.unsqueeze(1) < 0, correction,
        torch.zeros_like(correction))
    u_corrected = u_ref + correction
    return u_corrected

def simulate_reference_controller_unicycle(x0, tmax=10.0, dt=0.01, v
    =1.0, K=66.0):
```

```
83      N = int(tmax / dt)
84      traj = np.zeros((N + 1, 3))
85      traj[0] = x0.copy()
86      state = x0.copy()
87      for i in range(N):
88          state_t = torch.tensor(state, dtype=torch.float32, device=
                device).unsqueeze(0)
89          u_ref = simple_ref_controller(state_t, K=K)
90          u_val = u_ref.item()
91          psi = state[2]
92          dot_x = v * np.cos(psi)
93          dot_y = v * np.sin(psi)
94          dot_psi = u_val
95          xdot = np.array([dot_x, dot_y, dot_psi])
96          state = state + dt * xdot
97          traj[i + 1] = state
98      return traj
99
100 def simulate_barrier_controller_unicycle(x0, network, tmax=10.0, dt
        =0.01, v=1.0, K=66.0, alpha=1.0, kappa=10.0, barrier_scale=1.0):
101     N = int(tmax / dt)
102     traj = np.zeros((N + 1, 3))
103     traj[0] = x0.copy()
104     state = x0.copy()
105     for i in range(N):
106         state_t = torch.tensor(state, dtype=torch.float32, device=
                device).unsqueeze(0)
107         u_ref = simple_ref_controller(state_t, K=K)
108         if u_ref.dim() == 1:
109             u_ref = u_ref.unsqueeze(1)
110         u_bar = unicycle_barr_controller(state_t, network, u_ref=u_ref,
                alpha=alpha, kappa=kappa)
111         u_final = u_bar.item()
112         psi = state[2]
113         dot_x = v * np.cos(psi)
114         dot_y = v * np.sin(psi)
115         dot_psi = np.clip(u_final, -1, 1)
116         xdot = np.array([dot_x, dot_y, dot_psi])
117         state = state + dt * xdot
118         traj[i + 1] = state
119     return traj
120
121 if __name__ == "__main__":
122     x0 = np.array([-1.5, -1.5, 0.78], dtype=float)
123     Tfinal = 10.0
124     dt = 0.01
125     network = BarrierNetwork().to(device)
```

```
126     network.load_state_dict(torch.load("/content/
            barrier_network_unicycle_final_2025-02-12␣12_35_14.461340.pt",
            map_location=device))
127     network.eval()
128     traj_ref = simulate_reference_controller_unicycle(x0, tmax=Tfinal,
            dt=dt, v=1, K=50.0)
129     traj_bar = simulate_barrier_controller_unicycle(x0, network, tmax
            =0.39*Tfinal, dt=dt, v=1, K=50.0, alpha=2, kappa=5,
            barrier_scale=1)
130     plt.figure(figsize=(5, 5))
131     unsafe_rect = plt.Rectangle((-0.2, -0.2), 0.4, 0.4, linewidth=2,
            edgecolor='r', facecolor='none')
132     plt.gca().add_patch(unsafe_rect)
133     plt.xlim([-2, 2])
134     plt.ylim([-2, 2])
135     plt.plot(traj_ref[:, 0], traj_ref[:, 1], 'b', label='Reference␣Only
            ')
136     plt.plot(traj_bar[:, 0], traj_bar[:, 1], 'm', label='Reference␣+␣
            Barrier')
137     plt.plot(x0[0], x0[1], 'go', label='Start')
138     plt.plot(1.0, 1.0, 'rx', label='Goal')
139     plt.title("Unicycle␣Path")
140     plt.xlabel("x")
141     plt.ylabel("y")
142     plt.legend()
143     plt.tight_layout()
144     plt.show()
```

Listing A.2: Algorithm 2 Implementation for Unicycle

Bibliography

[1] Eitan Altman. *Constrained Markov decision processes*. Routledge, 2021.

[2] Joshua Achiam et al. "Constrained policy optimization". In: *International conference on machine learning*. PMLR. 2017, pp. 22–31.

[3] Somil Bansal et al. "Hamilton-Jacobi reachability: A brief overview and recent advances". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. 2017. DOI: 10.1109/CDC.2017.8263977.

[4] Manan Tayal et al. "A Physics-Informed Machine Learning Framework for Safe and Optimal Control of Autonomous Systems". In: *arXiv preprint arXiv:2502.11057* (2025).

[5] Aditya Singh, Zeyuan Feng, and Somil Bansal. *Exact Imposition of Safety Boundary Conditions in Neural Reachable Tubes*. 2024. arXiv: 2404.00814 [cs.RO]. URL: https://arxiv.org/abs/2404.00814.

[6] Aaron D Ames, Jessy W Grizzle, and Paulo Tabuada. "Control barrier function based quadratic programs with application to adaptive cruise control". In: *53rd IEEE Conference on Decision and Control*. IEEE. 2014, pp. 6271–6278.

[7] Aaron D. Ames et al. "Control Barrier Function Based Quadratic Programs for Safety Critical Systems". In: *IEEE Transactions on Automatic Control* 62.8 (2017), pp. 3861–3876. DOI: 10.1109/tac.2016.2638961.

[8] Guofan Wu and Koushil Sreenath. "Safety-critical control of a planar quadrotor". In: *2016 American Control Conference (ACC)*. 2016, pp. 2252–2258. DOI: 10.1109/ACC.2016.7525253.

[9] Manan Tayal et al. "Control barrier functions in dynamic uavs for kinematic obstacle avoidance: A collision cone approach". In: *2024 American Control Conference (ACC)*. IEEE. 2024, pp. 3722–3727.

[10] Aaron D Ames et al. "Control barrier functions: Theory and applications". In: *18th European control conference (ECC)*. IEEE. 2019, pp. 3420–3431.

[11] Manan Tayal and Shishir Kolathaya. "Safe Legged Locomotion using Collision Cone Control Barrier Functions (C3BFs)". In: *arXiv preprint arXiv:2309.01898* (2023).

[12] A. Papachristodoulou and S. Prajna. "A tutorial on sum of squares techniques for systems analysis". In: *Proceedings of the 2005, American Control Conference, 2005.* 2005, 2686–2700 vol. 4. DOI: 10.1109/ACC.2005.1470374.

[13] Ufuk Topcu, Andrew Packard, and Peter Seiler. "Local stability analysis using simulations and sum-of-squares programming". In: *Automatica* 44.10 (2008), pp. 2669–2675. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2008.03.010.

[14] Alexander Robey et al. "Learning Control Barrier Functions from Expert Demonstrations". In: *2020 59th IEEE Conference on Decision and Control (CDC)*. 2020, pp. 3717–3724. DOI: 10.1109/CDC42340.2020.9303785.

[15] Hengjun Zhao et al. "Synthesizing barrier certificates using neural networks". In: *Proceedings of the 23rd international conference on hybrid systems: Computation and control*. 2020, pp. 1–11.

[16] Alessandro Abate et al. "Fossil: A software tool for the formal synthesis of Lyapunov functions and barrier certificates using neural networks". In: *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*. 2021, pp. 1–11.

[17] Qingye Zhao et al. "Verifying Neural Network Controlled Systems Using Neural Networks". In: *25th ACM International Conference on Hybrid Systems: Computation and Control*. 2022, pp. 1–11.

[18] Hongchao Zhang et al. "Exact Verification of ReLU Neural Control Barrier Functions". In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh et al. Vol. 36. Curran Associates, Inc., 2023, pp. 5685–5705.

[19] Charles Dawson, Sicun Gao, and Chuchu Fan. "Safe Control With Learned Certificates: A Survey of Neural Lyapunov, Barrier, and Contraction Methods for Robotics and Control". In: *IEEE Transactions on Robotics* (2023).

[20] Simin Liu, Changliu Liu, and John Dolan. "Safe control under input limits with neural control barrier functions". In: *Conference on Robot Learning*. PMLR. 2023, pp. 1970–1980.

[21] Manan Tayal et al. "CP-NCBF: A Conformal Prediction-based Approach to Synthesize Verified Neural Control Barrier Functions". In: *arXiv preprint arXiv:2503.17395* (2025).

[22] Bolun Dai, Prashanth Krishnamurthy, and Farshad Khorrami. "Learning a Better Control Barrier Function". In: *2022 IEEE 61st Conference on Decision and Control (CDC)*. 2022, pp. 945–950. DOI: 10.1109/CDC51059.2022.9993334.

[23] Manan Tayal et al. "Learning a Formally Verified Control Barrier Function in Stochastic Environment". In: *2024 IEEE 63rd Conference on Decision and Control (CDC)*. 2024, pp. 4098–4104. DOI: 10.1109/CDC56724.2024.10886052.

[24] Manan Tayal et al. "Semi-Supervised Safe Visuomotor Policy Synthesis using Barrier Certificates". In: *arXiv preprint arXiv:2409.12616* (2024).

[25] Fabio Camilli, Lars Grüne, and Fabian Wirth. "Control Lyapunov Functions and Zubov's Method". In: *SIAM J. Control and Optimization* 47 (Jan. 2008), pp. 301–326. DOI: 10.1137/06065129X.

[26] Jun Liu et al. *Towards Learning and Verifying Maximal Neural Lyapunov Functions*. 2023. arXiv: 2304.07215 [math.OC]. URL: https://arxiv.org/abs/2304.07215.

[27] Jun Liu et al. *Formally Verified Physics-Informed Neural Control Lyapunov Functions*. 2024. arXiv: 2409.20528 [eess.SY]. URL: https://arxiv.org/abs/2409.20528.

[28] Jun Liu et al. *Physics-Informed Neural Network Lyapunov Functions: PDE Characterization, Learning, and Verification*. 2025. arXiv: 2312.09131 [math.OC]. URL: https://arxiv.org/abs/2312.09131.

[29] Eduardo Sontag. "Sontag, E.D.: A 'universal' construction of Artstein's theorem on nonlinear stabilization. System and Control Letters 13(2), 117-123". In: *Systems and Control Letters* 13 (Aug. 1989), pp. 117–123. DOI: 10.1016/0167-6911(89)90028-5.

[30] Aaron D. Ames et al. "Rapidly Exponentially Stabilizing Control Lyapunov Functions and Hybrid Zero Dynamics". In: *IEEE Transactions on Automatic Control* 59.4 (2014), pp. 876–891. DOI: 10.1109/TAC.2014.2299335.

[31] Kevin Galloway et al. "Torque Saturation in Bipedal Robotic Walking Through Control Lyapunov Function-Based Quadratic Programs". In: *IEEE Access* 3 (2015), pp. 323–332. DOI: 10.1109/ACCESS.2015.2419630.

[32] Liyun Dai et al. *Barrier Certificates Revisited*. 2013. arXiv: 1310.6481 [cs.SY]. URL: https://arxiv.org/abs/1310.6481.

[33] Hui Kong et al. *Exponential-Condition-Based Barrier Certificate Generation for Safety Verification of Hybrid Systems*. 2013. arXiv: 1303.6885 [cs.SE]. URL: https://arxiv.org/abs/1303.6885.

[34] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[35] Xiangru Xu et al. "Robustness of Control Barrier Functions for Safety Critical Control**This work is partially supported by the National Science Foundation Grants 1239055, 1239037 and 1239085." In: *IFAC-PapersOnLine* 48.27 (2015). Analysis and Design of Hybrid Systems ADHS, pp. 54–61. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2015.11.152. URL: https://www.sciencedirect.com/science/article/pii/S2405896315024106.

[36] F. Blanchini. "Set invariance in control". In: *Automatica* 35.11 (1999), pp. 1747–1767. ISSN: 0005-1098. DOI: https://doi.org/10.1016/S0005-1098(99)00113-2. URL: https://www.sciencedirect.com/science/article/pii/S0005109899001132.

[37] Shao-Chen Hsu, Xiangru Xu, and Aaron D. Ames. "Control barrier function based quadratic programs with application to bipedal robotic walking". In: *2015 American Control Conference (ACC)*. Funding by NSF. IEEE, 2015, pp. 4542–4548. ISBN: 978-1-4799-8684-2. DOI: 10.1109/ACC.2015.7172044.

[38] Fabio Camilli, Lars Grüne, and Fabian Wirth. "Control Lyapunov Functions and Zubov's Method". In: *SIAM Journal on Control and Optimization* 47.1 (2008), pp. 301–326. DOI: 10.1137/06065129X. eprint: https://doi.org/10.1137/06065129X. URL: https://doi.org/10.1137/06065129X.

[39] Jun Liu. "Converse Barrier Functions via Lyapunov Functions". In: *IEEE Transactions on Automatic Control* 67.1 (2022), pp. 497–503. DOI: 10.1109/TAC.2021.3085419.

[40] V. I. Zubov. *Methods of A. M. Lyapunov and Their Applications*. English. Retrieved from Archive.org on 2025-04-01. 1964. URL: https://archive.org/details/v.-i.-zubov-methods-of-a.-m.-lyapunov-and-their-applications-1964.

[41] B. Siciliano et al. *Robotics: Modelling, Planning and Control*. Springer Series in Advanced Textbooks in Control and Signal Processing. 632 pages. Springer, 2009.

[42] G. Dicaro. *Feedback Based Control IV*. https://web2.qatar.cmu.edu/~gdicaro/16311-Fall17/slides/16311-12-FeedbackBased-Control-IV.pdf. Slides. Accessed: 2025-04-13. 2017.

[43] Hongchao Zhang et al. "Fault Tolerant Neural Control Barrier Functions for Robotic Systems under Sensor Faults and Attacks". In: *arXiv preprint arXiv:2402.18677* (2024).

[44] Unitree Robotics. *Unitree Go1 – Intelligent Bionic Quadruped Robot Companion*. Accessed: 2025-04-13. 2025. URL: https://www.unitree.com/go1.

[45] Cookie Robotics. *052*. https://cookierobotics.com/052/. Accessed: 2025-04-13.

[46] Mohammad Aali and Jun Liu. *Multiple Control Barrier Functions: An Application to Reactive Obstacle Avoidance for a Multi-steering Tractor-trailer System*. 2022. arXiv: 2209.05156 [eess.SY]. URL: https://arxiv.org/abs/2209.05156.

[47] Joseph Breeden and Dimitra Panagou. *Compositions of Multiple Control Barrier Functions Under Input Constraints*. 2023. arXiv: 2210.01354 [math.OC]. URL: https://arxiv.org/abs/2210.01354.