Proceedings of Machine Learning Research vol vvv:1–19, 2025

Opt-ODENet: A Neural ODE Framework with Differentiable QP Layers for Safe and Stable Control Design (longer version)

Keyan Miao KEYAN.MIAO@ENG.OX.AC.UK
Liqun Zhao LIQUN.ZHAO@ENG.OX.AC.UK
Han Wang HAN.WANG@ENG.OX.AC.UK
University of Oxford
Konstantinos Gatsis K.GATSIS@SOTON.AC.UK
University of Southampton
Antonis Papachristodoulou ANTONIS@ENG.OX.AC.UK
University of Oxford

Abstract

Designing controllers that achieve task objectives while ensuring safety is a key challenge in control systems. This work introduces Opt-ODENet, a Neural ODE framework with a differentiable Quadratic Programming (QP) optimization layer to enforce constraints as hard requirements. Eliminating the reliance on nominal controllers or large datasets, our framework solves the optimal control problem directly using Neural ODEs. Stability and convergence are ensured through Control Lyapunov Functions (CLFs) in the loss function, while Control Barrier Functions (CBFs) embedded in the QP layer enforce real-time safety. By integrating the differentiable QP layer with Neural ODEs, we demonstrate compatibility with the adjoint method for gradient computation, enabling the learning of the CBF class-$\mathcal{K}$ function and control network parameters. Experiments validate its effectiveness in balancing safety and performance.

Keywords: Neural ODEs, differentiable optimization layer, CLF, CBF, safe control

1. Introduction

Designing safe and efficient control systems is crucial in automation and control. Controllers must accomplish tasks while ensuring safety in environments under strict constraints. While learning-based control has shown promise in complementing traditional methods, integrating safety guarantees with task performance remains challenging. In this work, we propose a learning-based control framework that combines Neural Ordinary Differential Equations (Neural ODEs) with Control Barrier Functions (CBFs) and Control Lyapunov Functions (CLFs) to address this challenge.

Neural ODEs (Chen et al., 2018), provide a natural framework for modeling and optimizing continuous-time dynamic behaviors by treating networks as systems governed by differential equations. They offer adaptability, flexibility, and smooth gradient computation, making them well-suited for applications such as trajectory planning, state observer design, and system identification (Liang et al., 2021; Miao and Gatsis, 2023; Djeumou et al., 2022; Böttcher et al., 2022). However, ensuring task completion and safety in real-world applications requires additional mechanisms.

CBFs (Ames et al., 2019, 2016) and CLFs (Ames et al., 2014) are widely used in control theory to enforce safety constraints and ensure stability by defining safe sets and guaranteeing convergence to target states, respectively. Recently, CBFs and CLFs have been incorporated into learning-based frameworks, where they are often used as soft constraints to guide control design (Jin et al., 2020;

Zhao et al., 2024) or to improve performance in machine learning tasks such as image classification (Miao and Gatsis, 2024). Furthermore, the co-design of controllers with these functions has enabled adaptive and robust control in dynamic environments (Wang et al., 2023, 2024; Dawson et al., 2022). To enforce constraints as hard requirements, we incorporate Quadratic Programming (QP) into a learning framework with differentiable optimization layer (Amos and Kolter, 2017). QP embeds CBFs to enforce real-time safety constraints, while CLFs are incorporated into the loss function to ensure stability and task performance without conflicting with hard constraints. Its differentiability enables gradient-based optimization, allowing learning parameters of class $\mathcal{K}$ function for CBF constraints and preceding network parameters.

Several works closely related to learning-based control with safety considerations have been proposed, yet they exhibit certain limitations. Some rely on soft constraints to handle safety, which may fail in critical scenarios or only consider stability properties(Sandoval et al., 2023; Ip et al., 2024). Others utilize fixed CBF parameters, limiting their adaptability to varying environments (Pereira et al., 2021). Certain approaches depend on a nominal controller to guide control (Taylor et al., 2020), with some adopting supervised learning that relies on nominal safe controllers or reference trajectories and require large sampled datasets (Xiao et al., 2023b,a; Bachhuber et al., 2023). Furthermore, some works focus primarily on motion planning, adjusting abstract signals to satisfy constraints rather than directly controlling physical input signals (Nawaz et al., 2024).

The main contributions of this work are as follows: 1) We propose Opt-ODENet, a Neural ODE framework integrating differentiable QP layers for safe and stable control, eliminating the need for a nominal controller and reducing reliance on pre-designed policies or large datasets. 2) We incorporate CLFs into the loss function for stability and convergence, and embed CBFs as hard constraints into a differentiable QP layer, demonstrating compatibility with Neural ODEs and gradient computation via the adjoint method. 3) The framework adaptively learns the class-$\mathcal{K}$ function for CBF, enabling a flexible balance between safety and task performance. 4) Through experiments, we validate the proposed framework and show the applicability of higher-order CBFs (HOCBFs) within our framework, extending its capabilities to more complex systems.

2. Preliminaries

2.1. Neural ODEs with input

Residual Networks (ResNet) (He et al., 2016), a milestone in deep learning, introduced skip connections that inspired viewing layers as discrete-time dynamical systems. Extending this idea, Neural ODEs (Chen et al., 2018) model dynamics in continuous time, described as: Formally, a Neural ODE system with input is described as:

Definition 1 (Neural ODE with input) *A Neural ODE with input is a system of the form*

$$\begin{cases} \dot{x}(t) = \mathcal{F}(t, x(t), u(t), \theta), & t \in \mathcal{S} \\ x(t_0) = x_0 \end{cases} \tag{1}$$

where $\mathcal{S} := [t_0, t_f]$ $(t_0, t_f \in \mathbb{R}^+)$ *is the depth domain and* $\mathcal{F}$ *is a neural network referred to as ODENet with parameter* θ*;* $u(t)$ *is the input at time* t*.*

The terminal state $x(t_f)$, obtained by solving the initial value problem (IVP), represents the evolved system state. In Neural ODEs, depth corresponds to continuous progression along the time domain,

with ResNet interpreted as a discretization using the Euler method.
In applications like image classification, Neural ODEs map an initial state $x(t_0)$ to a terminal state $x(t_f)$, e.g., a label, by solving the IVP: $(t_f) = x(t_0) + \int_{t_0}^{t_f} \mathcal{F}(t, x(t), u(t), \theta)dt$. Training involves minimizing a loss ℓ based on the terminal state $x(t_f)$, which corresponds to a *Mayer* optimal control problem. A more general loss distributed across the depth domain is expressed as: $\ell \coloneqq \Phi(x(t_f)) + \int_{t_0}^{t_f} \mathcal{L}(x(t), u(t), \theta)dt$, where latent states evolve through layers to generate outputs. This training process can be formulated as the *Bolza* optimal control problem which can be solved recursively via gradient descent (GD). :

$$\begin{aligned} \min_{\theta \in U} \quad & \ell \\ s.t. \quad & \dot{x}(t) = \mathcal{F}(t, x(t), u(t), \theta),\ t \in \mathcal{S}, \\ & x(t_0) = x_0, \end{aligned} \tag{2}$$

2.2. CLF and CBF

Consider the dynamical system

$$\dot{x}(t) = \mathcal{F}(x(t), u(t)), \tag{3}$$

where $\mathcal{F}(x(t), u(t)) : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is a locally Lipschitz continuous function and $\mathcal{X}$ and $\mathcal{U}$ are compact sets. The controller $u(t)$ is designed for two objectives, namely ensuring *stability* and *safety*. The concept of safety involves an additional set $\mathcal{B} \subseteq \mathcal{X}$ which interprets certain physical requirements, such as collision avoidance for robots, lane keeping for vehicles, etc. Mathematically speaking, we say dynamical system (3) is *safe* if there exists a locally Lipschitz continuous controller $\tau(x(t)) : \mathcal{X} \to \mathcal{U}$ with $u(t) = \tau(x(t))$, such that $x(t) \in \mathcal{B}$ for any $x(0) \in \mathcal{B}$ and $t \geq 0$. One prevalent way to design a controller that ensures stability and safety relies on certificate functions. For stability, a Control Lyapunov Function (CLF) is defined as follows.

Definition 2 (Control Lyapunov Function) *Consider system (3). A locally positive definite and differentiable function $V(\cdot) : \mathcal{X} \to \mathbb{R}$ is called a Control Lyapunov Function if there exists a locally Lipschitz continuous controller $\tau(x(t)) : \mathcal{X} \to \mathcal{U}$, such that*

$$\frac{\partial V(x)}{\partial x}\mathcal{F}(x, \tau(x)) + \gamma(V(x)) \leq 0, \forall x \in \mathcal{X}, \tag{4}$$

where the function $\gamma(\cdot) : \mathbb{R}^+ \to \mathbb{R}$ is of class-$\mathcal{K}$, i.e. $\gamma(0) = 0$ and $\gamma(\cdot)$ is strictly increasing.

For safety, a Control Barrier Function (CBF) is defined as follows.

Definition 3 (Control Barrier Function (Ames et al., 2019)) *Consider system (3) and the safe set $\mathcal{B}$. A differentiable function $B(x)$ is called a Control Barrier Function (CBF) if $\mathcal{B} \coloneqq \{x \in \mathcal{X} : B(x) \geq 0\}$, and there exists a locally Lipschitz continuous controller $\tau(x(t)) : \mathcal{X} \to \mathcal{U}$, such that*

$$\frac{\partial B(x)}{\partial x}\mathcal{F}(x, \tau(x)) + \alpha(B(x)) \geq 0, \forall x \in \mathcal{B}. \tag{5}$$

The function $\alpha(\cdot) : \mathbb{R} \to \mathbb{R}$ is of extended class-$\mathcal{K}$.

2.3. Differentiable Optimization Layer

A differentiable optimization problem refers to a class of optimization problems whose solutions can be differentiated through backpropagation. This feature allows such optimization problems to function as layers within deep learning architectures, enabling the encoding of constraints and complex dependencies that traditional convolutional and fully connected layers typically cannot capture. In this work, we incorporate the differentiable optimization layer OptNet proposed by Amos and Kolter (2017) to implement the CBF-QP layer to incorporate the safety constraint.
OptNet defined a neural network layer based on quadratic programming problem:

$$\begin{aligned} z_{i+1} &= \arg\min_{z} \frac{1}{2} z^T Q(z_i) z + q(z_i) z^T \\ s.t. \quad & A(z_i) z = b(z_i), G(z_i) z \leq h(z_i) \end{aligned} \tag{6}$$

where $Q \in \mathbb{R}^{n \times n}$, $q \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $G \in \mathbb{R}^{p \times n}$, $h \in \mathbb{R}^p$ and $Q \geq 0$. z_i is the output of the previous layer, z_{i+1} is the output of the optimization layer.
The forward pass of the layer consists of formulating and solving the optimization problem. More critically, the backward pass requires computing the derivatives of the QP solution with respect to its input parameters, which are obtained by differentiating the Karush-Kuhn-Tucker (KKT) conditions.

3. Problem Formulation

Consider a control-affine system

$$\dot{x} = f(x) + g(x)u \tag{7}$$

where x is the state, u is the control input, and the locally Lipschitz continuous functions $f(x)$, $g(x)$ are known. Furthermore, we assume a CBF $B(x)$ that defines the safe set $\mathcal{B} := \{x : B(x) \geq 0\}$ is known *a priori*. This is a reasonable assumption since $f(x)$ and $g(x)$ are both known and time-invariant. Our goal is to design a state-feedback control policy $u(t) = \tau(x(t))$, parameterized by θ, that satisfies the following requirements:

(1) Task achievement: Stabilizes the system (7) around a desired terminal state $\hat{x}$ from an arbitrary initial point x_0.
(2) Constraint satisfaction: Ensures safety (CBF constraints) of the closed-loop system $\dot{x} = f(x) + g(x)\tau(x)$.
(3) Performance and Safety trade-off: Co-optimizes the control performance while achieving a minimum safety margin along the trajectory.

By achieving a minimum safety margin, we mean that the system should avoid aggressively approaching the boundary of the safe set $\mathcal{B}$ to optimize the control performance, such as minimizing stabilization time.
Motivated by the requirements, we propose the following safety-constrained optimal control problem:

$$\begin{aligned} \min_{\theta=(\theta_1,\theta_2)\in U} \quad & \ell = \Phi\left(x\left(t_f\right), \hat{x}\right) + \int_{t_0}^{t_f} \mathcal{L}\left(x\left(t\right), u\left(t\right), \theta\right) dt \\ \text{s.t.} \quad & \dot{x}\left(t\right) = \mathcal{F}(x,u) = f(x) + g(x)u, \quad t \in \mathcal{S} \\ & \dot{B}(x(t)) + \alpha(\theta_2) B(x(t)) \geq 0, \quad t \in \mathcal{S} \\ & x\left(t_0\right) = x_0, \quad u(t) = \tau(x, \theta_1, \theta_2) \end{aligned} \tag{8}$$

Here, $\theta = (\theta_1, \theta_2)$ represents the parameters of the problem, with θ_1 parameterizing the control network and θ_2 characterizing the class-$\mathcal{K}$ function $\alpha(\cdot)$ for the CBF constraints.

4. Proposed Approach

Solving problem (8) is challenging not only because of the optimal control nature, but also due to the hard safety constraints. To address these difficulties, we propose a learning-based control framework that effectively leverages Neural ODEs and differentiable optimization layers. The framework consists of the following components:

- Neural ODE Dynamics: The system dynamics and the control policy are parameterized using a Neural ODE $\dot{x} = \mathcal{F}(x, \tau(x, \theta))$ to allow for continuous-time modeling.
- Differentiable CBF-QP Layer: Safety constraints are enforced through a CBF, embedded within a differentiable QP layer.

$$\begin{aligned} u_{safe} = \quad & \arg\min_u \quad \frac{1}{2} u^T Q(u_{nom}) u + q(u_{nom}) u^T \\ & \text{s.t.} \qquad Au = b, Gu \leq h \end{aligned} \tag{9}$$

  where in our problem, $Q = I$, $q = u_{nom}$, A and b are absent, $G = -\frac{\partial B(x)}{\partial x} g(x)$ and $h = \frac{\partial B(x)}{\partial x} f(x) + \alpha(B(x), \theta_2)$. $u_{nom} = \pi(x, \theta_1)$ where π represents the controller neural network with parameter θ_1, and $u_{safe} = \tau(u_{nom}, \theta_2) = \tau(x, \theta_1, \theta_2)$ represents the output of the CBF-QP layer (9) with the parameter θ_2. This allows real-time satisfaction of the safety requirements by dynamically modulating the control input. Additionally, the QP layer enables learning of the function $\alpha(\cdot)$, providing greater flexibility in enforcing safety constraints.
- Training Objective: The control policy $\tau(x, \theta)$ is trained to minimize a composite cost function ℓ as in (8), which incorporates a CLF-based loss to stabilize the system while embedding CBFs as hard constraints within the QP layer to enforce safety. The differentiable framework enables gradient-based optimization while maintaining compliance with safety constraints.

The proposed framework is illustrated in Figure 1, which outlines the Neural ODE-based control architecture integrating a differentiable CBF-QP layer for safety enforcement.

Opt-ODENet: Gradients Computation In the following, we show how to use the adjoint method to train the proposed framework in a memory efficient way.

Proposition 4 *Consider the Neural ODE-based framework 1 and the loss function ℓ. Let θ_1 be the parameter of the controller neural network $\tau(\cdot)$, and θ_2 the parameter of the differentiable CBF-QP layer. Define $\mu_1(t_0)$ and $\mu_2(t_0)$ by the gradient of loss ℓ to θ_1 and θ_2:*

$$\mu_1(t_0) := \nabla_{\theta_1} \ell, \quad \mu_2(t_0) := \nabla_{\theta_2} \ell. \tag{10}$$

Then, the gradients μ_1 and μ_2 can be updated by the following differential equations with boundary conditions:

$$\begin{aligned} \dot{x}(t) &= \mathcal{F}(x(t), \tau(x, \theta_1, \theta_2)), \quad x(t_0) = x_0, \\ \dot{p}(t) &= -p \frac{\partial \mathcal{F}}{\partial x} - \frac{\partial \mathcal{L}}{\partial x}, \quad p(t_f) = \frac{\partial \Phi}{\partial x(t_f)}, \\ \dot{\mu}_1(t) &= -p \frac{\partial \mathcal{F}}{\partial \theta_1} - \frac{\partial \mathcal{L}}{\partial \theta_1}, \quad \mu_1(t_f) = \mathbb{0}_{n_{\theta_1}}, \\ \dot{\mu}_2(t) &= -p \frac{\partial \mathcal{F}}{\partial \theta_2} - \frac{\partial \mathcal{L}}{\partial \theta_2}, \quad \mu_2(t_f) = \mathbb{0}_{n_{\theta_2}}, \end{aligned} \tag{11}$$

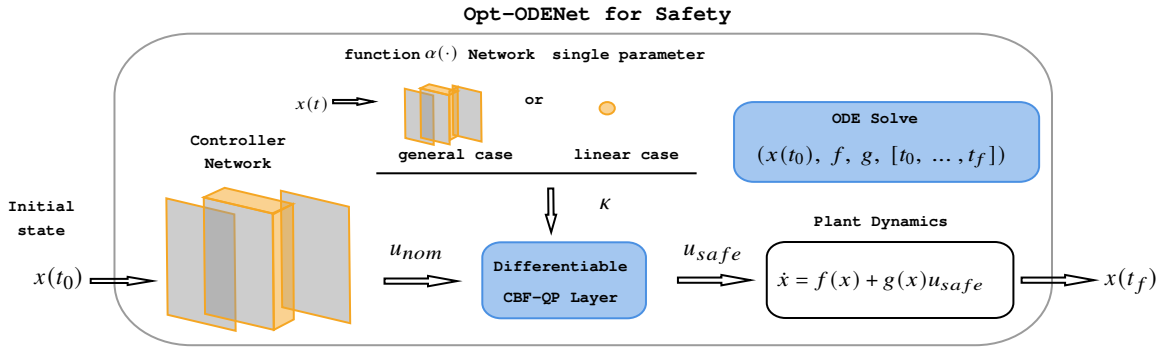


Figure 1: Schematics of the Neural ODE-based controller with a differentiable CBF-QP layer enforcing safety constraints

The differential equations (11) are derived by applying the adjoint method to an optimal control problem. In the following proof, we demonstrate the gradient computation for Neural ODEs with an embedded differentiable QP layer, leveraging the interconnected structure of the controller network. Due to space limitations, more details of the proof can be found in the supplementary material.

Proof Consider (7); we have that

$$\frac{\partial \mathcal{F}}{\partial x} = \frac{\partial f}{\partial x} + \frac{\partial g}{\partial x}\tau(x,\theta_1,\theta_2) + g\frac{\partial \tau(x,\theta_1,\theta_2)}{\partial x}. \tag{12}$$

Using chain rule, $\frac{\partial \tau(x,\theta_1,\theta_2)}{\partial x}$ is equal to

$$\frac{\partial \tau(x,\theta_1,\theta_2)}{\partial x} = \frac{\partial \tau(x,\theta_1,\theta_2)}{\partial \pi(x,\theta_1)}\frac{\partial \pi(x,\theta_1)}{\partial x}, \tag{13}$$

where $\frac{\partial \pi(x,\theta_1)}{\partial x}$ can be calculated by Automatic Differentiation (AD). We then calculate the first term by considering the first-order optimality condition of the differentiable CBF-QP layer (9). Consider the general case of QP problem where where all components A, b, G, h, Q, and q are present, and Q is positive definite, A and G have full rank . The Lagrangian of (9) is given by

$$L(u,\nu,\lambda) = \frac{1}{2}u^T Q u + q^T u + \nu^T(Au-b) + \lambda^T(Gu-h) \tag{14}$$

where ν are the dual variables on the equality constraints and $\lambda \geq 0$ are the dual variables on the inequality constraints. The KKT conditions for stationary, primal feasibility, and complementary slacknesss are

$$\begin{aligned} Qu^* + q + A^T\nu^* + G^T\lambda^* &= 0 \\ Au^* - b &= 0 \\ D(\lambda^*)(Gu^* - h) &= 0, \end{aligned} \tag{15}$$

where $D(\cdot)$ is a diagonal matrix with entries those of the vector argument, and u^*, ν^* and λ^* are the optimal primal and dual variables. Taking the differentials of these conditions gives the equations

$$\begin{bmatrix} Q & A^T & G^T \\ A & 0 & 0 \\ D(\lambda^*)G & 0 & D(Gu^*-h) \end{bmatrix} \begin{bmatrix} du \\ d\nu \\ d\lambda \end{bmatrix} = -\begin{bmatrix} dQu^* + dq + dG^T\lambda^* + dA^T\nu^* \\ dAu^* - db \\ D(\lambda^*)dGu^* - D(\lambda^*)dh \end{bmatrix} \tag{16}$$

First, according to (15), we can have an implicit function F (Dontchev and Rockafellar, 2009):

$$F(X,Y) = \begin{bmatrix} Qu^* + q + A^T\nu^* + G^T\lambda^* \\ Au^* - b \\ D(\lambda^*)(Gu^* - h) \end{bmatrix} = 0 \tag{17}$$

where $X = [Q; q; A; b; G; h]$, $Y = [u; \nu; \lambda]$. Hence, the Jacobian with respect to Y is:

$$\begin{aligned} J_{F,Y} &= \left[\begin{array}{c|c|c} J_{F,u} & J_{F,\nu} & J_{F,\lambda} \end{array}\right] \\ &= \left[\begin{array}{c|c|c} Q & A^T & G^T \\ A & 0 & 0 \\ D(\lambda^*)G & 0 & D(Gu^* - h) \end{array}\right] \end{aligned} \tag{18}$$

note that all the vectors are column vectors. Next, we calculate the Jacobian with respect to X (note that all the matrices and vectors are operated as column vectors):

$$\begin{aligned} J_{F,X} &= \left[\begin{array}{c|c|c|c|c|c} J_{F,Q} & J_{F,q} & J_{F,A} & J_{F,b} & J_{F,G} & J_{F,h} \end{array}\right] \\ &= \left[\begin{array}{c|c|c|c|c|c} I \otimes u^{*T} & I & \nu^{*T} \otimes I & 0 & \lambda^{*T} \otimes I & 0 \\ 0 & 0 & I \otimes u^{*T} & -I & 0 & 0 \\ 0 & 0 & 0 & 0 & D(\lambda^*)I \otimes u^{*T} & -D(\lambda^*) \end{array}\right] \end{aligned} \tag{19}$$

where $\otimes$ represents Kronecker product. The Jacobian of Y with respect to X is then given by

$$J_{Y,X} = -[J_{F,Y}]^{-1}[J_{F,X}], \tag{20}$$

where $J_{F,Y}$ is given in (18), and $J_{F,X}$ is given in (19). Note in our problem, A is not present rather than being set to 0. This does not affect the invertibility of $J_{F,Y}$. $\frac{\partial \tau(x,\theta_1,\theta_2)}{\partial \pi(x,\theta_1)}$ can be obtained via $\frac{\partial \tau(x,\theta_1,\theta_2)}{\partial q}$ from $J_{Y,X}$. We then consider $\frac{\partial \mathcal{F}}{\partial \theta_2}$; we have that

$$\frac{\partial \mathcal{F}}{\partial \theta_2} = \frac{\partial \mathcal{F}}{\partial \tau(x,\theta_1,\theta_2)} \frac{\partial \tau(x,\theta_1,\theta_2)}{\partial \theta_2} = \underbrace{\frac{\partial \mathcal{F}}{\partial \tau(x,\theta_1,\theta_2)}}_{AD} \underbrace{\frac{\partial \tau(x,\theta_1,\theta_2)}{\partial h(x,\theta_2)}}_{(20)} \underbrace{\frac{\partial h(x,\theta_2)}{\partial \theta_2}}_{AD} \tag{21}$$

Similarly, for $\frac{\partial \mathcal{F}}{\partial \theta_1}$, it holds that

$$\frac{\partial \mathcal{F}}{\partial \theta_1} = \frac{\partial \mathcal{F}}{\partial \tau(x,\theta_1,\theta_2)} \frac{\partial \tau(x,\theta_1,\theta_2)}{\partial \theta_1} = \underbrace{\frac{\partial \mathcal{F}}{\partial \tau(x,\theta_1,\theta_2)}}_{AD} \underbrace{\frac{\partial \tau(x_1,\theta_1,\theta_2)}{\partial \pi(x,\theta_1)}}_{(20)} \underbrace{\frac{\partial \pi(x,\theta_1)}{\partial \theta_1}}_{AD}. \tag{22}$$

The remaining terms, $\frac{\partial \mathcal{L}}{\partial x}$, $\frac{\partial \mathcal{L}}{\partial \theta_1}$ and $\frac{\partial \mathcal{L}}{\partial \theta_2}$ can also be efficiently calculated via AD and (20) similarly as $\mathcal{L}$ is already an explicit function of x. ■

Control Objective: CLF Loss Inspired by LyaNet Rodriguez et al. (2022), for a given target and terminal loss Φ, the potential function V can be designed as

$$V_x(\cdot) \coloneqq \Phi\left(x\left(\cdot\right)\right) \tag{23}$$

Algorithm 1: Training Algorithm

Input: Sampled initial state $x(t_0)$, number of iterations $n > 0$, time steps $[t_0, t_1, \cdots, t_f]$, dynamics f and g, safety requirements

Output: A safe and stable controller u_{safe} and class-$\mathcal{K}$ function

Construct CBF and CLF;
while $i \leq n$ **do**
    $u_{nom} \leftarrow NN_{controller}(x, \theta)$;
    $u_{safe} \leftarrow QP(CBF, u_{nom}, \psi)$;
    $x \leftarrow ODESolver(x(t_0), f, g, u_{safe}, [t_0, \cdots, t_f])$;
    Compute loss ℓ by CLF over the trajectories;
    $\theta_1 \leftarrow Optimizer(\nabla_{\theta_1}\ell, \theta_1)$; `/* Update controller NN parameters */`
    $\theta_2 \leftarrow Optimizer(\nabla_{\theta_2}\ell, \theta_2)$; `/* Update function` $\alpha(\cdot)$ `parameters */`
end

Then a point-wise Lyapunov loss can be designed as

$$\mathcal{V} \coloneqq \max\left\{0, \frac{\partial V_x}{\partial x}\mathcal{F}(x,u) + \gamma\, V_x(x)\right\} \tag{24}$$

Equation (24) signifies the local violation of the invariance condition specified in (4). When $\mathcal{V} = 0$ holds for all data in the time interval, the inference dynamics exhibit exponential convergence towards a prediction that minimizes the loss. The Lyapunov loss for the dynamic system (7) is

$$\ell \coloneqq \mathbb{E}\left[\int_{t_0}^{t_f} \mathcal{V} dt\right] \tag{25}$$

Remark 5 (Rodriguez et al. (2022)) *Consider the Lyapunov loss above. If there exists a parameter θ^* of the dynamic system that satisfies $\ell(\theta^*) = 0$, then:*

- *The potential function V_x is an exponentially stabilizing Lyapunov function with θ^*;*
- *For $t \in [t_0, t_f]$, the dynamics satisfy the following convergence expression with respect to the loss Φ:*

$$\Phi(x(t)) \leq \Phi(x(t_0))\, e^{-\kappa t} \tag{26}$$

The Lyapunov method provides a guaranteed convergence rate for a broader range of problems and affords us the opportunity to manually select the rate of convergence, as there might be instances where too fast dynamics are not preferred.
Algorithm 1 demonstrates how the steps above are integrated in our proposed method.

5. Case Study

In this experiment setup, a unicycle is tasked with reaching the designated location, i.e., the destination, while avoiding collisions with an obstacle. The dynamics of the system are:

$$\dot{x}(t) = \begin{bmatrix} \cos\theta(t) & 0 \\ \sin\theta(t) & 0 \\ 0 & 1.0 \end{bmatrix} u(t). \tag{27}$$

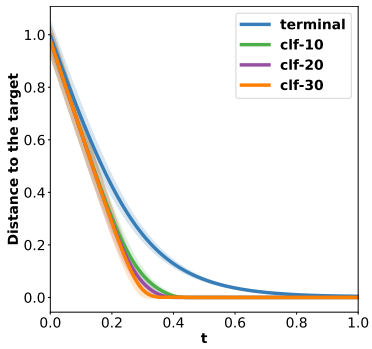


Figure 2: Distance over time with varying ℓ

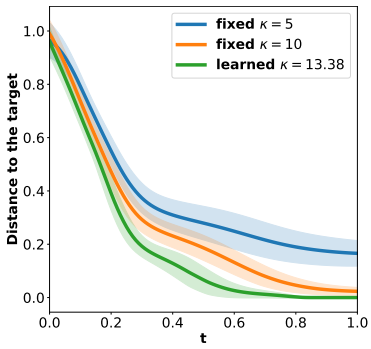
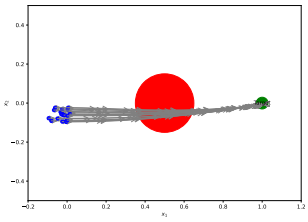
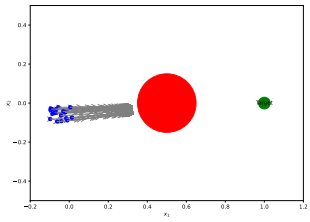


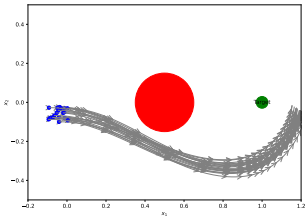
Figure 3: Distance over time with varying κ



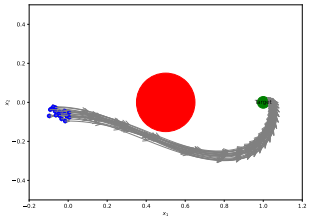
(*a*) without CBF-QP



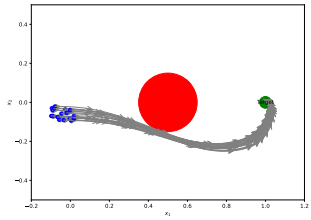
(*b*) CBF-QP only in Inference



(*c*) CBF-QP: fixed $\kappa = 5$



(*d*) CBF-QP: fixed $\kappa = 10$



(*e*) CBF-QP: learned $\kappa = 13.38$

Figure 4: Tested Trajectories under Different CBF-QP Settings

The control signal $u(t) = [v(t), \omega(t)]^T$ consists of linear and angular velocity inputs. A point at a distance l_p ahead of the unicycle is used to establish safety constraints for collision-free navigation, and the function $p : \mathbb{R}^3 \to \mathbb{R}^2$ is defined as:

$$p(x(t)) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + l_p \begin{bmatrix} \cos\theta(t) \\ \sin\theta(t) \end{bmatrix}. \tag{28}$$

In this case, the CBF is defined as $h(x(t)) = \frac{1}{2}\left((p(x(t)) - p_{\text{obs}})^2 - \delta_1^2\right)$, where p_{obs} represents the position of the obstacle, and δ_1 denotes the minimum safe distance between the unicycle and the obstacle. For $\alpha(\cdot)$, we adopt a linear form where only the linear coefficient κ is learned, as illustrated in Figure 1. The CLF is defined as $V(x(t)) = \frac{1}{2}\left((p(x(t)) - p_{\text{tar}})^2 - \delta_2^2\right)$, where p_{tar} represents the position of the target, and δ_2 specifies the radius of the target region.

First, we compared the system's performance when only a terminal loss was used versus when a CLF-based loss was incorporated. The trajectories were evaluated by tracking the evolution of the system's distance to the target over time, as shown in Figure 2. The curve with the CLF (with varying γ) converges significantly faster to the target compared to the one with only terminal loss as in Böttcher et al. (2022), highlighting the advantage of CLF in accelerating convergence.

Next, we analyzed the role of the CBF-QP layer in ensuring safety and achieving the task. Several

	fixed κ=5	fixed κ = 10	learned κ = 13.38	No CBF-QP	CBF-QP only in Inference
Mean Error	0.3623	0.2544	0.2036	0.1587	0.7088
Collision	No	No	No	Yes	No

Table 1: Comparison of Performance under Different Settings

Figure 5: Training results on Simulated Cars with HOCBF

configurations were tested, including no QP layer as in Ip et al. (2024), applying the QP layer only during inference (i.e., without being involved during controller training), and using fixed or learned $\alpha(\cdot)$ parameters as in Pereira et al. (2021) (in our case, the parameter is the linear coefficient κ).
The results, shown in Figure 4, reveal significant differences in performance. Without the QP layer, the system fails to avoid obstacles and results in collisions. When the QP layer is applied only during inference, safety is maintained (in contrast, the method in Sandoval et al. (2023) using soft constraints lacks theoretical safety guarantees), but the system loses its ability to effectively approach the target. For fixed $\kappa = 5$ and $\kappa = 10$, the method either leads to overly conservative behaviour, preventing task completion, or fails to strike a balance between safety and performance. In contrast, learning κ during training enables the system to adaptively balance safety and task objectives.
Figure 3 shows the evolution of the distance to the target over time with different coefficient κ. The presence of obstacles naturally slow convergence due to safety constraints compared to Figure 2. However, learning κ achieves a better trade-off, ensuring collision-free navigation while maintaining effective convergence. Table 1 reports the mean error and collision counts, showing that without the CBF-QP layer, collisions occur, inference-only QP sacrifices performance, and fixed κ is either too conservative or fails. The learned κ achieves minimal error with no collisions.
We also extend our framework to the High Order CBF (HOCBF) (Xiao and Belta, 2019) case in the Simulated Cars environment, with Figure 5 presenting the training results. The results demonstrate that the system successfully converges to task objectives while satisfying safety constraints. Additional comparisons, analysis and experiments details are shown in the supplementary material.

6. Discussions and Future work

Future work will focus on several key areas. First, designing more general and effective class-$\mathcal{K}$ functions for the CBF can improve the framework's flexibility and performance. Second, optimizing the computational efficiency of the QP layer is crucial for enabling real-time applications. Third, extending the framework to tackle more complex tasks, such as handling moving obstacles, will further validate its robustness. Finally, exploring learning-based methods to directly design the CBF could lead to a fully adaptive and scalable control framework.

Acknowledgments

The authors would like to express sincere gratitude to the Engineering and Physical Sciences Research Council (EPSRC) for the financial support provided through the grant EP/T517811/1, and AP through the EEBio Programme Grant EP/Y014073/1.

References

Aaron D Ames, Kevin Galloway, Koushil Sreenath, and Jessy W Grizzle. Rapidly exponentially stabilizing control lyapunov functions and hybrid zero dynamics. *IEEE Transactions on Automatic Control*, 59(4):876–891, 2014.

Aaron D Ames, Xiangru Xu, Jessy W Grizzle, and Paulo Tabuada. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control*, 62(8): 3861–3876, 2016.

Aaron D Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *2019 18th European control conference (ECC)*, pages 3420–3431. IEEE, 2019.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International conference on machine learning*, pages 136–145. PMLR, 2017.

Simon Bachhuber, Ive Weygers, and Thomas Seel. Neural odes for data-driven automatic self-design of finite-time output feedback control for unknown nonlinear dynamics. *IEEE Control Systems Letters*, 2023.

Lucas Böttcher, Nino Antulov-Fantulin, and Thomas Asikis. Ai pontryagin or how artificial neural networks learn to control dynamical systems. *Nature communications*, 13(1):333, 2022.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Charles Dawson, Zengyi Qin, Sicun Gao, and Chuchu Fan. Safe nonlinear control using robust neural lyapunov-barrier functions. In *Conference on Robot Learning*, pages 1724–1735. PMLR, 2022.

Franck Djeumou, Cyrus Neary, Eric Goubault, Sylvie Putot, and Ufuk Topcu. Neural networks with physics-informed architectures and constraints for dynamical systems modeling. In *Learning for Dynamics and Control Conference*, pages 263–277. PMLR, 2022.

Asen L Dontchev and R Tyrrell Rockafellar. *Implicit functions and solution mappings*, volume 543. Springer, 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Joshua Hang Sai Ip, Georgios Makrygiorgos, and Ali Mesbah. Lyapunov neural ode feedback control policies. *arXiv preprint arXiv:2409.00393*, 2024.

Wanxin Jin, Zhaoran Wang, Zhuoran Yang, and Shaoshuai Mou. Neural certificates for safe control policies. *arXiv preprint arXiv:2006.08465*, 2020.

Yuxuan Liang, Kun Ouyang, Hanshu Yan, Yiwei Wang, Zekun Tong, and Roger Zimmermann. Modeling trajectories with neural ordinary differential equations. In *IJCAI*, pages 1498–1504, 2021.

Keyan Miao and Konstantinos Gatsis. Learning robust state observers using neural odes. In *Learning for Dynamics and Control Conference*, pages 208–219. PMLR, 2023.

Keyan Miao and Konstantinos Gatsis. How deep do we need: Accelerating training and inference of neural ODEs via control perspective. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*. PMLR, 21–27 Jul 2024.

Farhad Nawaz, Tianyu Li, Nikolai Matni, and Nadia Figueroa. Learning complex motion plans using neural odes with safety and stability guarantees. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 17216–17222, 2024. doi: 10.1109/ICRA57147.2024.10611584.

Marcus Pereira, Ziyi Wang, Ioannis Exarchos, and Evangelos Theodorou. Safe optimal control using stochastic barrier functions and deep forward-backward sdes. In *Conference on Robot Learning*, pages 1783–1801. PMLR, 2021.

Ivan Dario Jimenez Rodriguez, Aaron Ames, and Yisong Yue. Lyanet: A lyapunov framework for training neural odes. In *International Conference on Machine Learning*, pages 18687–18703. PMLR, 2022.

Ilya Orson Sandoval, Panagiotis Petsagkourakis, and Ehecatl Antonio del Rio-Chanona. Neural odes as feedback policies for nonlinear optimal control. *IFAC-PapersOnLine*, 56(2):4816–4821, 2023.

Andrew Taylor, Andrew Singletary, Yisong Yue, and Aaron Ames. Learning for safety-critical control with control barrier functions. In *Learning for Dynamics and Control*, pages 708–717. PMLR, 2020.

Han Wang, Kostas Margellos, and Antonis Papachristodoulou. Safety verification and controller synthesis for systems with input constraints. *IFAC-PapersOnLine*, 56(2):1698–1703, 2023.

Han Wang, Kostas Margellos, Antonis Papachristodoulou, and Claudio De Persis. Convex co-design of control barrier function and safe feedback controller under input constraints. *arXiv preprint arXiv:2403.11763*, 2024.

Wei Xiao and Calin Belta. Control barrier functions for systems with high relative degree. In *2019 IEEE 58th conference on decision and control (CDC)*, pages 474–479. IEEE, 2019.

Wei Xiao, Ross Allen, and Daniela Rus. Safe neural control for non-affine control systems with differentiable control barrier functions. In *2023 62nd IEEE Conference on Decision and Control (CDC)*, pages 3366–3371. IEEE, 2023a.

Wei Xiao, Tsun-Hsuan Wang, Ramin Hasani, Makram Chahine, Alexander Amini, Xiao Li, and Daniela Rus. Barriernet: Differentiable control barrier functions for learning of safe robot control. *IEEE Transactions on Robotics*, 39(3):2289–2307, 2023b.

Liqun Zhao, Keyan Miao, Konstantinos Gatsis, and Antonis Papachristodoulou. Nlbac: A neural ordinary differential equations-based framework for stable and safe reinforcement learning. *arXiv preprint arXiv:2401.13148*, 2024.

7. Appendix

7.1. Gradients Computation of Opt-ODENet

7.1.1. Adjoint Method for Back-propagation in Neural ODEs

A Neural ODE is a system of the form

$$\dot{x}\left(t\right)=\mathcal{F}\left(t,x\left(t\right),\theta\right)\quad t\in\mathcal{S} \tag{29}$$

where $\mathcal{S}:=\left[t_0,t_f\right]$ $(t_0,t_f\in\mathbb{R}^+)$ is the depth domain and $\mathcal{F}$ is a neural network called ODENet which is chosen as part of the machine learning model with parameter θ.
Loss:

$$\ell:=\Phi\left(x\left(t_f\right)\right)+\int_{t_0}^{t_f}\mathcal{L}\left(x\left(t\right),\theta,t\right)dt \tag{30}$$

Consider Problem (29)-(30). The gradient of loss ℓ with respect to parameter θ is

$$\nabla_\theta\ell=\mu\left(t_0\right) \tag{31}$$

where $x\left(t\right)$, $p\left(t\right)$ and $\mu\left(t\right)$ satisfy the boundary value problem:

$$\begin{aligned}
&\dot{x}\left(t\right)=\mathcal{F},x\left(t_0\right)=x_0\\
&\dot{p}\left(t\right)=-p\left(t\right)\frac{\partial\mathcal{F}}{\partial x}-\frac{\partial\mathcal{L}}{\partial z},\ p\left(t_f\right)=\frac{\partial\Phi}{\partial x\left(t_f\right)}\\
&\dot{\mu}\left(t\right)=-p\left(t\right)\frac{\partial\mathcal{F}}{\partial\theta}-\frac{\partial\mathcal{L}}{\partial\theta},\ \mu\left(t_f\right)=\mathbb{0}_{n_\theta}
\end{aligned} \tag{32}$$

Proof Define the augmented objective function with a Lagrange multiplier p as

$$L=\ell-\int_{t_0}^{t_f}p(t)\left[\dot{x}(t)-\mathcal{F}\left(t,x(t),\theta\right)\right]dt \tag{33}$$

and $\dot{x}-\mathcal{F}=0$ always holds by construction, so $p(t)$ can be freely assigned while $\frac{dL}{d\theta}=\frac{d\ell}{d\theta}$. As to the integration part on right hand side of (33), we have

$$\begin{aligned}
\int_{t_0}^{t_f}p(t)\left(\dot{x}-\mathcal{F}\right)dt=&\ p(t)x(t)\Big|_{t_0}^{t_f}\\
&-\int_{t_0}^{t_f}\dot{p}(t)x(t)dt-\int_{t_0}^{t_f}p(t)\mathcal{F}dt\\
&=p\left(t_f\right)x\left(t_f\right)\\
&-p\left(t_0\right)x\left(t_0\right)-\int_{t_0}^{t_f}\left(\dot{p}(t)x(t)+p(t)\mathcal{F}\right)dt
\end{aligned}$$

Hence,

$$L = \Phi\left(x\left(t_f\right)\right) - p\left(t_f\right)x\left(t_f\right) + p\left(t_0\right)x\left(t_0\right) + \int_{t_0}^{t_f}\left(\dot{p}(t)x(t) + p(t)\mathcal{F} + \mathcal{L}\right)dt$$

Then the gradient of ℓ with respect to θ can be computed as

$$\begin{aligned}\frac{d\ell}{d\theta} = \frac{dL}{d\theta} &= \left(\frac{\partial\Phi}{\partial x\left(t_f\right)} - p(t_f)\right)\frac{dx(t_f)}{d\theta} \\ &+ \int_{t_0}^{t_f}\left(\dot{p}(t)\frac{dx(t)}{d\theta} + p(t)\left(\frac{\partial\mathcal{F}}{\partial\theta} + \frac{\partial\mathcal{F}}{\partial x}\frac{dx}{d\theta}\right) + \frac{\partial\mathcal{L}}{\partial\theta} + \frac{\partial\mathcal{L}}{\partial x}\frac{dx}{d\theta}\right)dt\end{aligned} \tag{34}$$

Now if we set this Lagrange multiplier as the adjoint state for the Hamiltonian function

$$H\left(x, p, \theta\right) = p\mathcal{F} + \mathcal{L} \tag{35}$$

and according to PMP, the optimality conditions requires

$$\dot{x} = \frac{\partial H}{\partial p} = \mathcal{F},\ \dot{p} = -\frac{\partial H}{\partial x} = -p\frac{\partial\mathcal{F}}{\partial zx} - \frac{\partial\mathcal{L}}{\partial x} \tag{36}$$

with initial conditions $x(t_0) = x_0$ and $p(t_f) = \frac{\partial\Phi}{\partial x(t_f)} = \frac{\partial\Phi}{\partial x\left(t_f\right)}$. Substituting (36) into (34), it can be obtained that

$$\begin{aligned}\frac{d\ell}{d\theta} &= \int_{t_0}^{t_f}\left(p(t)\frac{\partial\mathcal{F}}{\partial\theta} + \frac{\partial L}{\partial\theta}\right)dt \\ &= \int_{t_f}^{t_0}\left(-p(t)\frac{\partial\mathcal{F}}{\partial\theta} - \frac{\partial\mathcal{L}}{\partial\theta}\right)dt = \int_{t_f}^{t_0} -\frac{\partial H}{\partial\theta}dt\end{aligned} \tag{37}$$

proving the result. ■

7.1.2. GRADIENTS THROUGH QP LAYER

For a differentiable CBF-QP layer (9), the gradients of loss with respect to different variables through QP layer can be computed as follows.

Theorem 6 (Implicit Function Theorem) *Given three open setx $X \subseteq \mathbb{R}^n$, $Y \subseteq \mathbb{R}^m$ and $Z \subseteq \mathbb{R}^m$, if function $F : X \times Y \mapsto Z$ is continuously differentiable, and $(\hat{x}, \hat{y}) \in \mathbb{R}^n \times \mathbb{R}^m$ is a point for which*

$$F(\hat{x}, \hat{y}) = \hat{z}, \tag{38}$$

and the Jacobian of F with respect to $y \subseteq Y$

$$J_{F,y}\Big|_{i,j} = \left[\frac{\partial F_i}{\partial y_j}\right] \tag{39}$$

is invertible at $(\hat{x}, \hat{y})$, then there exists an open set $W \in \mathbb{R}^n$ with $x \in W$ and a unique continuously differentiable function $\phi : W \mapsto Y$ such that $y = \phi(x)$ and

$$F(x, y) = \hat{z} \tag{40}$$

holds for $x \in W$.
In addition, it can be shown that the partial derivatives of ϕ and W are given by

$$J_{y,x} = -\left[J_{F,y}\right]^{-1}\left[J_{F,x}\right]. \tag{41}$$

The Lagrangian of (9) is given by

$$L(u,\nu,\lambda) = \frac{1}{2}u^T Q u + q^T u + \nu^T(Au-b) + \lambda^T(Gu-h) \tag{42}$$

where ν are the dual variables on the equality constraints and $\lambda \geq 0$ are the dual variables on the inequality constraints. The KKT conditions for stationary, primal feasibility, and complementary slacknesss are

$$\begin{aligned} Qu^* + q + A^T\nu^* + G^T\lambda^* &= 0 \\ Au^* - b &= 0 \\ D(\lambda^*)(Gu^* - h) &= 0, \end{aligned} \tag{43}$$

where $D(\cdot)$ is a diagonal matrix with entries those of the vector argument, and u^*, ν^* and λ^* are the optimal primal and dual variables. Taking the differentials of these conditions gives the equations

$$\begin{aligned} dQu^* + Qdu + dq + dA^T\nu^* + A^T d\nu + dG^T\lambda^* + G^T d\lambda &= 0 \\ dAu^* + Adu - db &= 0 \\ D(Gu^* - h)d\lambda + D(\lambda^*)(dGu^* + Gdu - dh) &= 0, \end{aligned} \tag{44}$$

or written more compactly in matrix form

$$\begin{bmatrix} Q & A^T & G^T \\ A & 0 & 0 \\ D(\lambda^*)G & 0 & D(Gu^*-h) \end{bmatrix} \begin{bmatrix} du \\ d\nu \\ d\lambda \end{bmatrix} = -\begin{bmatrix} dQu^* + dq + dG^T\lambda^* + dA^T\nu^* \\ dAu^* - db \\ D(\lambda^*)dGu^* - D(\lambda^*)dh \end{bmatrix} \tag{45}$$

First, according to (43), we can have an implicit function F (Dontchev and Rockafellar, 2009):

$$F(X,Y) = \begin{bmatrix} Qu^* + q + A^T\nu^* + G^T\lambda^* \\ Au^* - b \\ D(\lambda^*)(Gu^* - h) \end{bmatrix} = 0 \tag{46}$$

where $X = [Q; q; A; b; G; h]$, $Y = [u; \nu; \lambda]$. Hence, the Jacobian with respect to Y is:

$$\begin{aligned} J_{F,Y} &= \left[\begin{array}{c|c|c} J_{F,u} & J_{F,\nu} & J_{F,\lambda} \end{array}\right] \\ &= \left[\begin{array}{c|c|c} Q & A^T & G^T \\ A & 0 & 0 \\ D(\lambda^*)G & 0 & D(Gu^*-h) \end{array}\right] \end{aligned} \tag{47}$$

note that all the vectors are column vectors. Next, we calculate the Jacobian with respect to X (note that all the matrices and vectors are operated as column vectors):

$$\begin{aligned} J_{F,X} &= \left[\begin{array}{c|c|c|c|c|c} J_{F,Q} & J_{F,q} & J_{F,A} & J_{F,b} & J_{F,G} & J_{F,h} \end{array}\right] \\ &= \left[\begin{array}{c|c|c|c|c|c} I \otimes u^{*T} & I & \nu^{*T} \otimes I & 0 & \lambda^{*T} \otimes I & 0 \\ 0 & 0 & I \otimes u^{*T} & -I & 0 & 0 \\ 0 & 0 & 0 & 0 & D(\lambda^*)I \otimes u^{*T} & -D(\lambda^*) \end{array}\right] \end{aligned} \tag{48}$$

where $\otimes$ represents Kronecker product. According to the Implicit Function Theorem 6, we obtain

$$\begin{aligned}\left(\frac{\partial l}{\partial X}\right)^T &= \left(\frac{\partial l}{\partial Y}\right)^T J_{Y,X} \\ &= -\left(\frac{\partial l}{\partial Y}\right)^T \left[J_{F,Y}\right]^{-1} \left[J_{F,X}\right]\end{aligned} \tag{49}$$

Since primal and dual variables λ^* and ν^* will not be passed into following layers of networks, we have $\left(\frac{\partial l}{\partial Y}\right)^T = \left[\left(\frac{\partial l}{\partial u^*}\right)^T, 0, 0\right]$ where 0 on the right hand side are zero vectors with the same size as ν^{*T} and λ^{*T}. We define $\begin{bmatrix} d_u \\ d_\nu \\ d_\lambda \end{bmatrix} = \left[J_{F,Y}^T\right]^{-1} \frac{\partial l}{\partial Y}$. According to the Implicit Function Theorem, we obtain $\frac{\partial l}{\partial X} = \left[J_{F,X}\right]^T \begin{bmatrix} d_u \\ d_\nu \\ d_\lambda \end{bmatrix}$, then

$$\begin{aligned}\frac{\partial l}{\partial X} &= \left[\begin{array}{c|c|c|c|c|c} I \otimes u^{*T} & I & \nu^{*T} \otimes I & 0 & \lambda^{*T} \otimes I & 0 \\ 0 & 0 & I \otimes u^{*T} & -I & 0 & 0 \\ 0 & 0 & 0 & 0 & D(\lambda^*) I \otimes u^{*T} & -D(\lambda^*) \end{array}\right]^T \begin{bmatrix} d_u \\ d_\nu \\ d_\lambda \end{bmatrix} \\ &= \begin{bmatrix} I \otimes u^* & 0 & 0 \\ I & 0 & 0 \\ \nu^* \otimes I & I \otimes u^* & 0 \\ 0 & -I & 0 \\ \lambda^* \otimes I & 0 & D(\lambda^*) I \otimes u^* \\ 0 & 0 & D(\lambda^*) \end{bmatrix} \begin{bmatrix} d_u \\ d_\nu \\ d_\lambda \end{bmatrix}\end{aligned} \tag{50}$$

Now, we can have

$$\begin{aligned} \frac{\partial l}{\partial Q} &= \frac{1}{2}\left(d_u u^{*T} + u^* d_u^T\right), \quad \frac{\partial l}{\partial q} = d_u \\ \frac{\partial l}{\partial A} &= \nu^* d_u^T + d_\nu u^{*T}, \quad \frac{\partial l}{\partial b} = -d_\nu \\ \frac{\partial l}{\partial G} &= D(\lambda^*) d_\lambda z^{*T} + \lambda^* d_u^T, \quad \frac{\partial l}{\partial h} = -D(\lambda^*) d_\lambda \end{aligned} \tag{51}$$

Or more directly, for example, we can obtain

$$\begin{aligned}\left(\frac{\partial l}{\partial q}\right)^T &= \left(\frac{\partial l}{\partial Y}\right)^T [J_{F,Y}]^{-1} [J_{F,q}] \\ &= \left(\frac{\partial l}{\partial u^*}\right)^T [J_{F,Y}]^{-1} \begin{bmatrix} I \\ 0 \\ 0 \end{bmatrix}\end{aligned} \tag{52}$$

If automatic differentiation method is used to compute the entire framework of Neural ODEs with QP layer, then Equation (51) represents the gradient of loss with respect to the variables through the QP layer.

7.1.3. GRADIENTS COMPUTATION OF OPT-ODENET (ADJOINT METHOD)

The results are shown in Proposition 4. As to $\frac{\partial \mathcal{L}}{\partial \theta_1}$ and $\frac{\partial \mathcal{L}}{\partial \theta_2}$, the results are as follows:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \theta_2} &= \frac{\partial \mathcal{L}}{\partial x}\frac{\partial x}{\partial \tau(x,\theta_1,\theta_2)}\frac{\partial \tau(x,\theta_1,\theta_2)}{\partial \theta_2} \\ &= \underbrace{\frac{\partial \mathcal{L}}{\partial x}}_{AD}\underbrace{\frac{\partial x}{\partial \tau(x,\theta_1,\theta_2)}}_{AD}\underbrace{\frac{\partial \tau(x,\theta_1,\theta_2)}{\partial h(x,\theta_2)}}_{(20)}\underbrace{\frac{\partial h(x,\theta_2)}{\partial \theta_2}}_{AD}\end{aligned} \tag{53}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \theta_1} &= \frac{\partial \mathcal{L}}{\partial x}\frac{\partial x}{\partial \tau(x,\theta_1,\theta_2)}\frac{\partial \tau(x,\theta_1,\theta_2)}{\partial \theta_1} \\ &= \underbrace{\frac{\partial \mathcal{L}}{\partial x}}_{AD}\underbrace{\frac{\partial x}{\partial \tau(x,\theta_1,\theta_2)}}_{AD}\underbrace{\frac{\partial \tau(x_1,\theta_1,\theta_2)}{\partial \pi(x,\theta_1)}}_{(20)}\underbrace{\frac{\partial \pi(x,\theta_1)}{\partial \theta_1}}_{AD}.\end{aligned} \tag{54}$$

7.2. Experiments

7.2.1. EXPERIMENTS DETAILS OF UNICYCLE

For the Unicycle environment, the controller network is designed as a two-layer neural network with a hidden state dimension of 64. The training is performed with a batch size of 32 over 100 epochs. The Neural ODE is solved using the Euler method, with a time interval of $[0, 1]$ and a step size of 0.01.

The setting in Equation (28) is also set up to deal with the different relative degrees for (27). Another way is modify the dynamics as follows:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\\ \dot{v}\end{bmatrix} = \begin{bmatrix} v\cos(\theta)\\ v\sin(\theta\\ 0\\ 0\end{bmatrix} + \begin{bmatrix}0 & 0\\ 0 & 0\\ 1.0 & 0\\ 0 & 1.0\end{bmatrix}\begin{bmatrix} v\\ \omega\end{bmatrix} \tag{55}$$

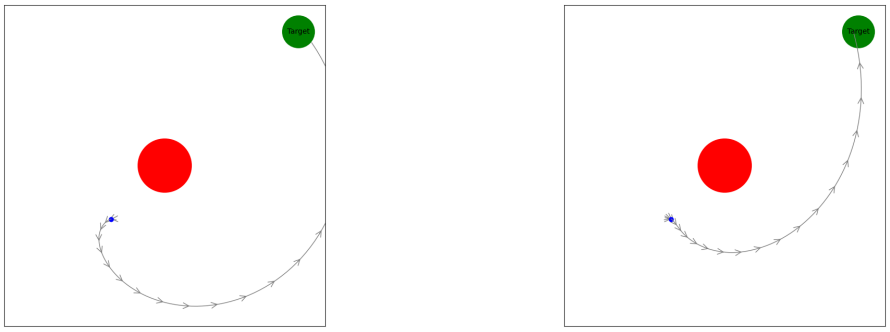
In this case, HOCBF should be used. We show some results to illustrate the importance of the choice of class-$\mathcal{K}$ function in Figure 7.2.1. In this case, a larger κ typically delays the system's response to CBF constraints, giving priority to the objective. This often results in the system adhering more closely to obstacles, as it adjusts later to safety constraints.

7.2.2. EXPERIMENTS DETAILS OF SIMULATED CARS

This environment, simulating a chain of five cars following each other on a straight road. The objective is to control the acceleration of the 4^{th} car to maintain a desired distance from the 3^{rd} car while avoiding collisions with other cars. The real dynamics of all cars except the 4^{th} one is given by:

$$\dot{x}_i(t) = \begin{bmatrix} v_i(t)\\ 0\end{bmatrix} + \begin{bmatrix} 0\\ 1+d_i\end{bmatrix} a_i(t) \qquad \forall i \in \{1,2,3,5\}.$$

Each state of the system is denoted as $x_i(t) = [p_i(t), v_i(t)]^T$, indicating the position $p_i(t)$ and velocity $v_i(t)$ of the i^{th} car at the time t, $d_i = 0.1$. The time interval used in this experiment is 0.02s.



(a) $\kappa = \{0.1, 0.1\}$ (b) $\kappa = \{1, 1\}$

Figure 6: Test Trajectories under Different HOCBF-QP Settings for Unicycle

The predefined velocity of the 1^{st} car is $v_s - 4\sin(t)$, where $v_s = 3.0$. Its acceleration is given as $a_1(t) = k_v(v_s - 4\sin(t) - v_1(t))$ where $k_v = 4.0$. Accelerations of Car 2 and 3 are given by:

$$a_i(t) = \begin{cases} k_v(v_s - v_i(t)) - k_b(p_{i-1}(t) - p_i(t)) & if\ |p_{i-1}(t) - p_i(t)| < 6.5 \\ k_v(v_s - v_i(t)) & otherwise, \end{cases}$$

where $k_b = 20.0$ and $i = 2, 3$. The acceleration of the 5^{th} car is:

$$a_5(t) = \begin{cases} k_v(v_s - v_5(t)) - k_b(p_3(t) - p_5(t)) & if\ |p_3(t) - p_5(t)| < 13.0 \\ k_v(v_s - v_5(t)) & otherwise. \end{cases}$$

The model of the 4^{th} car is as follows:

$$\dot{x}_4(t) = \begin{bmatrix} v_4(t) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1.0 \end{bmatrix} u(t),$$

where $u(t))$ is the acceleration of the 4^{th} car, and also the control signal generated by the controller at the time t.
$d(t) = p_3(t) - p_4(t)$ represents the distance between the 3^{rd} and 4^{th} car, and the objective is to let $d(t)$ fall within $[5.5, 6.5]$ as a desired region. Thus, the CLF is determined as $\|d(t) - d_{\text{desired}}\|$, where $d_{\text{desired}} = 6$. CBFs are defined as $h_1(x(t)) = p_3(t) - p_4(t) - \delta$ and $h_2(x(t)) = p_4(t) - p_5(t) - \delta$, with δ being the minimum required distance between the cars.
The controller network is designed as a four-layer neural network with hidden state dimensions of 128, 64, 32 and 12. The training is performed over 400 epochs. The Neural ODE is solved using the Euler method, with a time interval of $[0, 4]$ and a step size of 0.02.
The training results under different HOCBF-QP settings are shown in Figure 7 where the reward represents how often $d(t)$ falls with desired region. It can be found that when $\kappa = \{2, 1\}$, the whole training seems meaningless, the reason may be that the CBF cannot work normally and the car stucks when κ is too small and therefore contradicts with the whole setting, and CLF cannot help the car get out of this circumstance (the fourth car cannot monotonically going further from

Figure 7: Training results on Simulated Cars under Different HOCBF-QP Settings

both the third and the fifth car at the same time and therefore certain infeasibility happens, which makes the whole training problematic. - however in this case we can consider learning different κ for different safety constraints for improvement) When $\kappa = \{60, 900\}$, the safety constraints are valid and the safety violations are 0. For the reward, this case works worse than the case that κ is learned as $\{18.6, 96.04\}$. Hence, the effect of κ is not as straightforward as 'larger κ equals more aggressive behavior.' While a larger κ often delays the system's response to CBF constraints, allowing it to prioritize the CLF objective and potentially 'stick closer' to obstacles, this does not universally translate to faster or more efficient target convergence. The interaction between CBF and CLF objectives is context-dependent. For example, when the target lies near an obstacle, a larger κ might help the system avoid detours by adhering closely to the safety boundary. However, if the target direction naturally steers the system away from the obstacle, increasing κ adds little value and could even result in suboptimal trajectories due to delayed responses to other constraints. Thus, the impact of class-$\mathcal{K}$ function is dynamic and contextual, and in some cases it is important to learn it rather than fix it to better accommodate different CBF-CLF(objective) interactions, especially for hard constraints.