# Safe model-based Reinforcement Learning via Model Predictive Control and Control Barrier Functions

Kerim Dzhumageldyev

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

# Safe model-based Reinforcement Learning via Model Predictive Control and Control Barrier Functions

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Kerim Dzhumageldyev

October 10, 2025

Faculty of Mechanical Engineering (ME) · Delft University of Technology


# Abstract

Optimal control strategies are often combined with safety certificates to ensure both performance and safety in safety-critical systems. A prominent example is combining Model Predictive Control (MPC) with Control Barrier Functions (CBF). Yet, tuning MPC parameters and choosing an appropriate class $\mathcal{K}$ function in the CBF is challenging and problem dependent. This thesis introduces a safe model-based Reinforcement Learning (RL) framework where a parameterized MPC incorporates a CBF with a parameterized class $\mathcal{K}$ function and serves as a function approximator to learn improved safe control policies. Three variations are introduced, distinguished by the way the class $\mathcal{K}$ function is parameterized. The Learnable Optimal Decay CBF (LOPTD-CBF) extends the Optimal Decay CBF by allowing RL to tune the optimal decay parameters, improving performance while enhancing constraint feasibility and preserving safety guarantees. The Neural Network CBF (NN-CBF) parametrizes the decay term of a discrete exponential CBF with a neural network, enabling richer state-dependent safety conditions. Finally, the Recurrent Neural Network CBF (RNN-CBF) extends the NN-CBF with a recurrent architecture to handle time-varying CBF constraints, such as moving obstacles. Numerical experiments on a discrete double-integrator with static and dynamic obstacles demonstrate that the proposed methods improve performance while ensuring safety, each offering distinct trade-offs in performance, feasibility and complexity.

# Acknowledgments

Firstly, I would like to express my deepest gratitude to my supervisors, Azita and Filippo, for their immense support throughout this project. Azita, thank you for your invaluable guidance and insight. Our meetings always helped me gain clarity on the technical aspects of the project and shape its overall direction. Filippo, thank you so much for mentoring me all the way. Thank you for being so patient with all my ideas, even the stupid ones. I truly appreciate your great and detailed feedback on my reports and ideas, and I really enjoyed all our long discussions about the thesis. Sorry if I ended up taking too much of your time with them.

I would like to thank my friends for their support throughout all my years at Delft. It has been a joy to study, spend time, play sports and share countless moments together. A special thanks goes to my friends from high school who also came to Delft to study. My time here would not have been nearly as memorable without you.

I would also like to especially thank my girlfriend. Thank you for always being there for me throughout this entire process of writing, stressing and learning. I can't imagine that constantly asking you to read my work was much fun, but I truly appreciate your patience and support. I couldn't have done this without you.

Lastly, and most importantly, I would like to thank my family. I want to thank my brother for his continuous love, and support throughout the years. I also want to thank my parents for bringing me from Turkmenistan to the Netherlands and giving me the opportunity to grow up in a free and open society. Because of their courage and sacrifices, I was able to grow up in a country where I feel safe, can express my opinions freely and have access to quality education. This achievement is as much theirs as it is mine.

Delft, University of Technology — Kerim Dzhumageldyev
October 10, 2025

# Table of Contents

# Chapter 1

# Introduction

Safety is a concern at the core of numerous problems in control. Throughout the years, various methodologies and theoretical frameworks have been put forward to ensure safety in control systems. Among these, one of the most prominent methods in the current literature is the use of Control Barrier Functions (CBFs). In essence, CBFs ensure safety by requiring that system trajectories remain within a safe set defined by a barrier function usually denoted by $h(x)$. The idea is that any action taken by the system must guarantee that the state remains inside a control invariant set, meaning a set from which there always exists a control action that keeps the system inside it. To achieve this, the CBF regulates how fast the safety function $h(x)$ can change over time. Specifically, it imposes a lower bound on the allowable decrease of $h(x)$, governed by a class $\mathcal{K}$ function. The controller must then select an action that ensures the change in $h(x)$ does not drop below this bound [8, 10]. CBFs have been employed in a variety of safety-critical applications, such as automotive systems, multi-robot systems, quadrotors, and robotic systems [8, 1, 2, 7, 15, 23, 37].

Nevertheless, the pursuit of safety can sometimes conflict with the goal of a controller achieving optimal performance. While most optimization-based controllers focus on performance objectives, unless safety is explicitly incorporated, for instance through risk-averse optimization or safety filters, they do not inherently prioritize safe trajectories. In contrast, introducing safety filters enforces safer paths, but this often comes at the expense of reduced performance. One common approach is to combine CBFs with Quadratic Programming (QP) formulations, where the QP optimizes performance while enforcing safety through the CBF constraint [8, 10, 9, 2, 37]. Although this approach effectively balances safety and performance, it often leads to myopic policies. Even though these myopic policies still guarantee safe state trajectories due to the forward invariance condition of the CBF, they might become infeasible when other constraints are introduced, such as bounds on the control action. Additionally, these myopic policies have a greater difficulty balancing safety and performance, since they only look one step ahead, compared to policies with longer horizons.

In this context, the need to incorporate CBFs into optimization-based controllers with extended prediction horizons emerged to enable the evaluation of CBF conditions at future predicted states [36, 38]. This led to the integration of CBFs, as constraints, within the

Model Predictive Control (MPC) framework [25], thereby combining the predictive capabilities of MPC with the safety guarantees of CBFs.

Nevertheless, the integrated MPC-CBF framework often degrades nominal MPC performance because safety and performance objectives may conflict. The CBF constraint in the MPC shrinks the feasible set of states to guarantee safety, making it harder for the optimizer to find high-quality, or even feasible, trajectories that satisfy both safety and performance requirements over the entire horizon. This is especially apparent in cases when the MPC's tuning (cost weights, horizon, model) is suboptimal.

To address these challenges, Sabouni et al. introduced a novel idea [26] that extended the MPC-CBF framework by using model-free Deep Reinforcement Learning (DRL) to learn the objective weights of the MPC and a simple class $\mathcal{K}$ function of the CBF. The class $\mathcal{K}$ function, as introduced earlier, modulates the conservativeness of the CBF condition by restricting the rate at which trajectories may approach the safe-set boundary.

As an alternative, this thesis adopts an MPC-based Reinforcement Learning (RL) framework in which, unlike in the work of [26], the RL agent uses the MPC controller itself as its function approximator rather than a Neural Network (NN). The choice of using MPC-based RL is motivated by three main advantages. Firstly, the MPC inherently incorporates prior knowledge through the prediction model, as opposed to a DRL which is a black-box. Secondly, MPC controllers come with provable guarantees for stability and constraint satisfaction, whereas neural-network methods are largely uninterpretable. Moreover, although not straightforward, it is possible to preserve these theoretical properties of MPC (stability, recursive feasibility, and constraint guarantees) during the RL learning process [18]. Lastly, unlike neural networks and many other function approximation schemes, MPC directly handles input and state constraints in its optimization, ensuring they are always satisfied.

## 1-1 Problem Statement

The integration of MPC-based RL with CBFs remains largely unexplored. Therefore, the main research goal of this thesis is to explore this combination, with a specific focus on the trade-off between safety and performance in such a framework. This motivates the following main research question:

> To what extent can performance be improved while still guaranteeing safety in MPC-based Reinforcement Learning using Control Barrier Functions?

In addition to this main question, the research will be extended to investigate the following sub-questions:

1. *Which type of CBF formulation yields the best trade-off between safety and performance in MPC-based RL?*
   Different CBF formulations developed in this thesis will be implemented and tested within the MPC-based RL framework to identify the approach that provides the most favorable balance between safety guarantees and achievable performance.

2. *How does tuning the CBF condition affect the balance between safety and performance?*
   The CBF also plays an important role in regulating performance. To understand this role, the thesis investigates how much performance can be improved by tuning the CBF while still guaranteeing safety. In the numerical experiments, some cases will parametrize and learn only the CBF condition, while keeping all other learnable parameters fixed to assess this effect.

3. *How can MPC-based RL using CBFs guarantee safety in dynamic environments, e.g., multiple dynamic obstacles?*
   The framework will be extended to scenarios involving multiple moving obstacles, as commonly encountered in real-life safety applications, to investigate effective methods for addressing such complex situations.

## 1-2 Thesis Outline

To explore the goals outlined above, the thesis is structured as follows. Chapter 2 provides an overview of the theoretical background on MPC-based RL and CBF, which forms the basis for the methods developed later. Chapter 3 presents a literature review on existing data-driven techniques to learn CBFs as well as to improve the CBF condition. Chapter 4 then introduces the methods designed to address the research questions and the identified gap in the literature. Chapter 5 evaluates these methods through numerical experiments, first on a simple example and then on a more complex case. Finally, the thesis concludes with a summary of the results and recommendations for future research.

# Chapter 2

# Theoretical Background

This chapter presents the preliminary theoretical background required for the rest of the thesis. The first section reviews the fundamentals of MPC and RL to motivate the MPC-based RL framework. The second section introduces CBF theory and presents the various CBF formulations employed throughout this work.

## 2-1 MPC-based RL Framework

The integration of MPC with RL has emerged as a compelling approach to address the challenges of optimizing control policies in systems with known models. This section outlines the MPC-based RL framework, beginning with a brief introduction and review of the fundamental principles of MPC. Similarly, this chapter aims to provide a review of the core RL formulations, serving as contextualization for the MPC-based RL framework. Combining the ideas of RL and MPC, the MPC-based RL framework is introduced and presented in the form of a Q-learning algorithm.

### 2-1-1 Model Predictive Control

Model Predictive Control is a prominent optimal control strategy employed for dynamical systems when a model is known, especially in the presence of constraints. It uses a mathematical model to predict future state trajectories and optimize control actions over a set horizon [25]. At each time step, the optimization minimizes a cost function, often balancing desired performance and penalizing control effort while respecting constraints such as input, output, or state limits. Only the first control action from this optimized sequence is implemented, and the process is repeated with updated measurements, allowing MPC to adapt to system changes and disturbances. Mathematically, the MPC is formulated as follows.

$$\min_{X,U} \quad \ell_f(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k) \tag{2-1a}$$
$$\text{s.t.} \quad x_{k+1} = f(x_k, u_k), \quad k = 0, \ldots, N-1, \tag{2-1b}$$
$$x_0 = x_{\text{current}}, \tag{2-1c}$$
$$x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}, \quad k = 0, \ldots, N-1, \tag{2-1d}$$
$$x_N \in \mathcal{X}_f. \tag{2-1e}$$

Where the primal variables include the sequence of predicted states $X = \begin{bmatrix} x_0 \dots x_N \end{bmatrix}^\top$ and the sequence of predicted control inputs $U = \begin{bmatrix} u_0 \dots u_{N-1} \end{bmatrix}^\top$, over the prediction horizon $N$. The objective function minimizes the cumulative stage cost $\sum_{k=0}^{N-1} \ell(x_k, u_k)$ over the horizon and the terminal cost $\ell_f(x_N)$ at the final state $x_N$, where $\ell : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ and $\ell_f : \mathcal{X} \to \mathbb{R}$, with $\mathcal{X} \subseteq \mathbb{R}^n$ and $\mathcal{U} \subseteq \mathbb{R}^m$ denoting the admissible state and input sets, respectively. The prediction model of the system is defined by the dynamics $x_{k+1} = f(x_k, u_k)$, with $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ governing the evolution of the state $x_k$ under the control input $u_k$. The initial condition $x_0 = x_{\text{current}}$ denotes the measured current state of the actual system. The constraints in the MPC ensure that the state $x_k$ and the control input $u_k$ remain within the permissible sets $\mathcal{X}$ and $\mathcal{U}$, respectively, at each time step. The final state $x_N$ belongs to a terminal constraint set $\mathcal{X}_f$, the target state for the system. Moreover, as noted earlier, at each time step MPC solves this optimization problem and applies only the first optimal control input $u_0^\star$. This process is repeated in a receding-horizon fashion, allowing the MPC to account for new state measurements and disturbances. However, the advantages of MPC come at the cost of significant computational demands and the requirement for an accurate system model with well-tuned parameters, such as the prediction horizon and the weights of the cost function [25]. Consequently, there is a growing need to enhance MPC performance, even when the model is imprecise or the parameters are not optimally tuned.

## 2-2 Reinforcement Learning

Reinforcement Learning is a machine learning paradigm that focuses on training an agent to make sequential decisions by interacting with an environment in order to minimize a cumulative cost (or maximize a reward signal) [30]. The framework is typically modeled as a discrete-time Markov Decision Process, where transitions from state $s$ to $s'$ occur via action $a$ and are governed by the conditional probability density [5]

$$P[s' \mid s, a] : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1], \tag{2-2}$$

where $\mathcal{A} \subseteq \mathbb{R}^{n_a}$ is the action space and $\mathcal{S} \subseteq \mathbb{R}^{n_s}$ is the state space. The learning process within this framework is illustrated in Figure 2-1. At each time step, the agent observes the current state $s$, selects an action $a$ based on its policy, and incurs a cost $c$ from the environment, which then transitions to a new state $s'$. This cycle forms the foundation of RL, where the agent iteratively refines its policy by leveraging this feedback loop [30]. The objective of the

agent is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$, that minimizes the expected discounted cumulative stage cost:

$$J(\pi) \coloneqq \mathbb{E}\left[\sum_{k=0}^{T} \gamma_{\mathrm{RL}}^{k} L(s_k, \pi(s_k))\right], \tag{2-3}$$

where $L : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the stage cost, $\gamma_{\mathrm{RL}} \in (0, 1]$ is the discount factor and $T$ is the task length [5]. The optimal policy $\pi^\star$ is given then by

$$\pi^\star = \arg\min_{\pi} J(\pi). \tag{2-4}$$

An alternative way of learning the optimal policy is to use value functions, namely the state value function $V_\pi(s)$:

$$V_\pi(s) \;=\; \mathbb{E}\Big[\sum_{k=0}^{\infty} \gamma_{\mathrm{RL}}^{k} L(s_k, \pi(s_k)) \;\Big|\; s_0 = s\Big], \tag{2-5}$$

which gives the expected cost when starting in state $s$ and following policy $\pi(s)$, as well as the action-value function $Q_\pi(s, a)$:

$$Q_\pi(s, a) \;=\; \mathbb{E}\Big[L(s, a) \;+\; \sum_{k=1}^{\infty} \gamma_{\mathrm{RL}}^{k} L(s_k, \pi(s_k)) \;\Big|\; s_0 = s,\; a_0 = a\Big], \tag{2-6}$$

which measures the cost of taking action $a$ in state $s$ and then following policy $\pi$. Additionally, under a deterministic policy $Q(s, a)$ and $V(s)$ relate as

$$V(s) \;=\; Q\big(s, \pi(s)\big) \;=\; \min_{a \in \mathcal{A}} Q(s, a). \tag{2-7}$$

Finally, the optimal policy can be extracted using the optimal action-value function as follows:

$$\pi(s) \;=\; \arg\min_{a \in \mathcal{A}} Q(s, a). \tag{2-8}$$

In practice, it is impossible to find and characterize the true optimal policy $\pi^\star$ and the corresponding optimal value functions $\{V^\star, Q^\star\}$. To address this, RL relies on function approximation techniques, such as NNs or more recently MPC, which enable the formulation of approximate parametrized policies $\pi_\theta$ and approximate parametrized value functions $\{V_\theta, Q_\theta\}$[12, 18]. These parameters are learned either via policy-gradient methods, which update $\pi_\theta$ directly [27, 30], or via value-based methods, which estimate $\{V_\theta, Q_\theta\}$ and act greedily with resepct to $Q_\theta$. Depending on the algorithm (policy-gradient or value-based), gradients updates are performed either on $\pi_\theta$ parameters or $\{V_\theta, Q_\theta\}$ parameters via

$$\theta \leftarrow \theta - \eta \nabla_\theta \psi(s_k, a_k, s_{k+1}, \theta), \tag{2-9}$$

where $\psi(s_k, a_k, s_{k+1}, \theta)$ captures the performance of the controller, $\eta > 0$ is the learning rate, and $m$ is the batch size [5].

RL approaches can be further broadly classified into model-free and model-based methods [30]. Model-free RL relies solely on observed state transitions and incurred costs, without requiring explicit knowledge of the environment's dynamics. While this makes it relatively straightforward to implement, model-free RL is often sample-inefficient and requires extensive exploration for efficient learning. In contrast, model-based RL utilizes a learned or known

model of the environment to predict state transitions and incurred costs, enabling planning and more sample-efficient learning. However, model-based methods are sensitive to inaccuracies in the model, which can negatively impact the performance of the derived policy.

Despite its potential, RL faces several significant challenges. Most notably, RL algorithms often lack formal safety and stability guarantees, largely because they are commonly developed with neural networks, which are typically non-interpretable. This makes it challenging to ensure safety and stability in practical applications. Additionally, RL struggles to incorporate design considerations such as hard state and input constraints or structured stage and terminal costs. Instead, such considerations are typically handled indirectly through cost shaping or heuristics, which cannot provide hard guarantees on system behaviour [18].

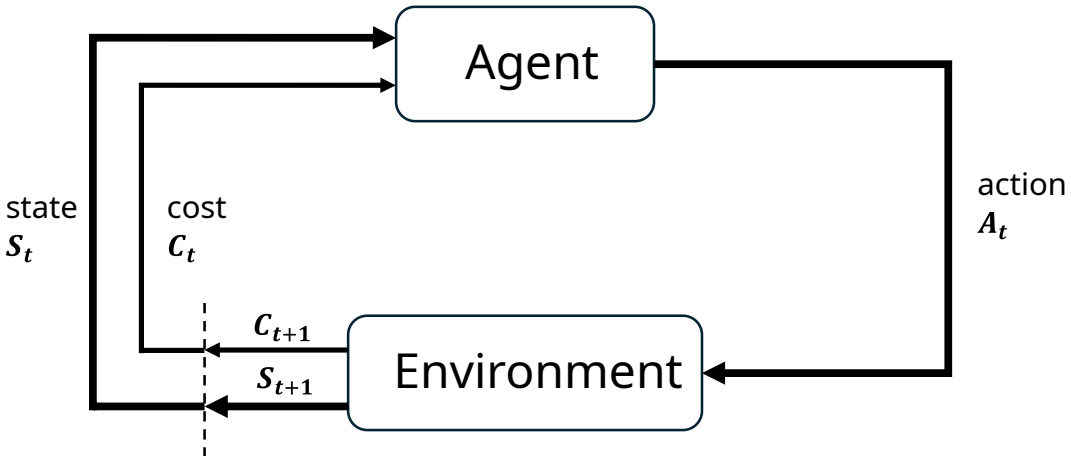



**Figure 2-1:** RL Framework: The agent–environment interaction, inspired by [30].

### 2-2-1 MPC-based RL framework

*Note.* Before introducing the MPC-based RL formulation, a short notational remark is made. Throughout this thesis, whenever MPC-based RL is used, the actual system state and action are denoted by $s$ and $a$, respectively. The MPC-predicted states and actions are denoted by $x_k$ and $u_k$, with $X = [x_0, \ldots, x_N]$ and $U = [u_0, \ldots, u_{N-1}]$ denoting the sequences of predicted states and actions. Since the methods proposed in this thesis rely on MPC-based RL, this convention is also consistently used in Chapters 4 and 5.

As discussed earlier, MPC excels at incorporating design considerations such as input constraints, performance via structured stage and terminal costs, as well as stability and safety guarantees into the control scheme. However, reliance on an accurate model is a significant drawback of MPC, especially when faced with model uncertainties. More importantly, proper tuning of MPC cost components can be tricky and time-consuming when carried out manually, especially in the presence of economic and/or nonconvex terms in the objective. Conversely, RL can learn a policy without an explicit model of the system, making it well-suited for environments with unknown or complex dynamics, as it only requires crafting a meaningful cost signal. Despite this advantage, RL algorithms struggle to provide formal guarantees on behavior, such as stability or constraint satisfaction. Furthermore, integrating specific design choices directly into the learning process remains a nontrivial task.

As a notational remark, throughout this thesis, whenever MPC-based RL is used, the actual system state and action are denoted by $s$ and $a$, respectively, while the MPC-predicted states and actions are denoted by $x_k$ and $u_k$, with $X = [x_0, \dots, x_N]$ and $U = [u_0, \dots, u_{N-1}]$. Since the methods proposed in this thesis rely on MPC-based RL, this convention is consistently used also in Chapters 4 and 5.

To leverage the strengths of both approaches, the MPC-based RL framework, originally proposed in [18], is introduced. In this framework, a parametric MPC acts as both a policy provider and a function approximator, while RL adjusts the MPC parameters to enhance performance. The framework can be visually represented in Figure 2-2 [5]. This diagram mirrors the generic RL architecture in Figure 2-1, but now explicitly embeds the parametric MPC as the policy approximator.

In this formulation, the parametric MPC is generally used to approximate the value function $V_\theta$, as defined in (2-10). Note that representing $V_\theta$ with MPC is not required, as alternative value-function approximators can be used within the MPC-based RL framework [18, 4].

$$
\begin{aligned}
V_\theta(s) = \min_{X,U,\Sigma} \quad & \lambda_\theta(x_0) + \gamma_{RL}^N(\ell_{f,\theta}(x_N) + w_f^\top \sigma_N) + \sum_{k=0}^{N-1} \gamma_{RL}^k(\ell_\theta(x_k, u_k) + w^\top \sigma_k) && \text{(2-10a)}\\
\text{s.t.} \quad & x_{k+1} = f_\theta(x_k, u_k), \quad k = 0, \dots, N-1 && \text{(2-10b)}\\
& x_0 = s, && \text{(2-10c)}\\
& g(u_k) \le 0, \quad k = 0, \dots, N-1 && \text{(2-10d)}\\
& c_\theta(x_k, u_k) \le \sigma_k, \quad k = 0, \dots, N-1 && \text{(2-10e)}\\
& c_\theta^f(x_N) \le \sigma_N. && \text{(2-10f)}
\end{aligned}
$$

Under this parameterization, $\lambda_\theta : \mathcal{S} \to \mathbb{R}$ represents an initial cost term. The stage cost $\ell_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ measures the cost associated with a given predicted state $x$ and a predicted input $u$, while the terminal cost $\ell_{f,\theta} : \mathcal{S} \to \mathbb{R}$ penalizes deviations from the desired terminal conditions [18, 4]. The system dynamics are modeled using the function $f_\theta : \mathcal{S} \times \mathcal{A} \to S$, which is a parameterized approximation of the true system dynamics. The function $g : \mathcal{A} \to \mathbb{R}$ represents input constraints, such as actuator limits. Inequality constraints are expressed as $c_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, for mixed state-input constraints and $c_\theta^f : \mathcal{S} \to \mathbb{R}$ for terminal constraints [18, 4]. Slack variables $\Sigma = \begin{bmatrix} \sigma_0 \dots \sigma_N \end{bmatrix}^\top$ are introduced to relax these constraints when necessary, ensuring feasibility. The weights $w$ and $w_f$ are assigned to these slack variables, controlling the degree of constraint relaxation by penalizing larger slack variables [18, 4]. Larger values of these weights render the constraints more rigid, closely approximating the behavior of an unrelaxed formulation, while smaller values provide greater flexibility by allowing the controlled violation of the original -inequalities, if needed. Additionally, the policy derived from the MPC always applies the first optimal control input ($u_0^\star$), from the computed sequence of control inputs, given as $\pi_\theta(s) = u_0^\star$. Similarly, the parametrized MPC action-value function $Q_\theta(s, a)$ can be defined as follows:

$$
\begin{aligned}
Q_\theta(s, a) = \min_{X,U,\Sigma} \quad & \text{(2-10a)} && \text{(2-11a)}\\
\text{s.t.} \quad & \text{(2-10b)} - \text{(2-10f)} && \text{(2-11b)}\\
& u_0 = a, && \text{(2-11c)}
\end{aligned}
$$

which resembles $V_\theta(s)$, with the addition of the input constraint $u_0 = a$ [18, 4]. The RL policy is then obtained by selecting the action that minimizes the learned action-value function, as shown before:

$$\pi_\theta(s) = \arg\min_a \quad Q_\theta(s, a). \tag{2-12a}$$

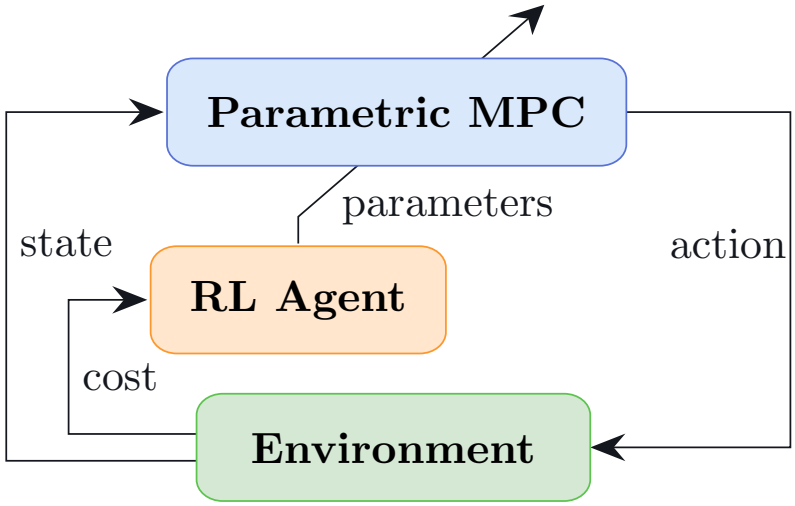



**Figure 2-2:** Diagram of the MPC-based RL architecture. [5]

### 2-2-2 Q-learning

The MPC-based RL framework can be adapted to a variety of RL algorithms. However, this thesis concentrates on Q-learning, a value-based method. The core idea of the Q-learning algortihm is to minimize the Bellman residual

$$\min_\theta \mathbb{E}\left[\|Q^\star(s, a) - Q_\theta(s, a)\|^2\right], \tag{2-13}$$

in order to learn the optimal action-value function and subsequently recover the optimal policy from it. In its recursive formulation, minimizing the Bellman residual amounts to driving the expected Temporal Difference (TD) error, defined as

$$\tau_k = L(s_k, a_k) + \gamma_{RL} \min_{a_{k+1}} Q_\theta(s_{k+1}, a_{k+1}) - Q_\theta(s_k, a_k), \tag{2-14}$$

to zero, where $\min_{a_{k+1}} Q_\theta(s_{k+1}, a_{k+1})$ reflects the greedy nature of Q-learning. This term ensures that the action selected at the next state minimizes the action-value estimate, prioritizing long-term cost reduction rather than strictly following the current policy. Alternatively, using the Bellman equation $V_\theta(s_k) = \min_{a_{k+1}} Q_\theta(s_{k+1}, a_{k+1})$, the TD error can also be written as

$$\tau_k = L(s_k, a_k) + \gamma_{RL} V_\theta(s_{k+1}) - Q_\theta(s_k, a_k). \tag{2-15}$$

Using (2-9) with $\psi(s_k, a_k, s_{k+1}, \theta) = \tau_k^2$ [5, 30] and differentiating $\psi(s_k, a_k, s_{k+1}, \theta)$ w.r.t $\theta$ under a semi-gradient (i.e. treating the target $L(s_k, a_k) + \gamma V_\theta(s_{k+1})$as constant) yields the parameter update

$$\theta \leftarrow \theta + \eta \tau_k \nabla_\theta Q_\theta(s_k, a_k), \tag{2-16}$$

where $\eta$ is the size of the update step [32, 18, 30].

### 2-2-3 Computing Sensitivities

To learn the MPC parameters using Q-learning, it is necessary to compute $\nabla_\theta Q_\theta(s_k, a_k)$ in (2-16), the sensitivities of the MPC solution with respect to $\theta$. To this end, the Lagrangian of the $Q_\theta(s, a)$ MPC problem is introduced [18]:

$$\begin{aligned}\mathcal{L}_\theta(s, a, y) = \lambda_\theta(x_0) + \gamma^N \ell_{f,\theta}(x_N) + \chi_0^\top (x_0 - s) + \mu_N^\top h_\theta^f(x_N) \\ + \sum_{k=0}^{N-1} \chi_{k+1}^\top \big(f_\theta(x_k, u_k) - x_{k+1}\big) + \nu_k^\top g_\theta(u_k) \\ + \gamma^k \ell_\theta(x_k, u_k) + \mu_k^\top h_\theta(x_k, u_k) + \zeta^\top (u_0 - a),\end{aligned} \tag{2-17}$$

where $\chi, \mu, \nu, \zeta$ are the Lagrange multipliers associated with the constraints of the action-value function, in equation (2-11) [18]. Let $y = (x, u, \chi, \mu, \nu, \zeta)$ denote the collection of primal–dual variables for the action-value problem. With the Lagrangian in place, the sensitivities of the action-value function are obtained as [18]:

$$\nabla_\theta Q_\theta(s, a) = \nabla_\theta \mathcal{L}_\theta(s, a, y^\star), \tag{2-18}$$

where $y^\star$ is a primal–dual optimal solution (KKT point).

Additionally, interior-point techniques are adopted for solving the MPCs in (2-10) and (2-11) to ensure that the gradients of inequality constraints remain well defined [18]. Accordingly, in this thesis the MPCs are solved using IPOPT, an interior-point optimizer [31].

## 2-3 Control Barrier Functions

Control Barrier Functions are powerful tools used as safety certificates to ensure that a system operates safely, by maintaining specific system states within a set of safe states. This section develops the CBF framework by first reviewing Nagumo's theorem, which provides the boundary condition used to establish forward invariance. Next, Lyapunov and Control Lyapunov functions (CLFs) are revisited to highlight the parallel with CBFs and to motivate CBFs as a safety-oriented counterpart to CLFs. With these preliminaries in place, the core CBF condition is presented together with its integration into CBF-based controllers. Exponential CBFs are introduced next and serve as the primary formulation used for the methods developed in the thesis. The section concludes with the Optimal-Decay CBF, which is central to one of the proposed methods. Throughout this section, standard control notation $x$ and $u$ is used to denote the system state and input, respectively.

### 2-3-1 Nagumo's Theorem

The safety guarantee given by the CBF is achieved through the concept of forward invariance [8, 10]. Forward invariance entails that for a given dynamical system and a set $\mathcal{C} \subseteq \mathbb{R}^n$, any trajectory of the system that starts inside the set $\mathcal{C}$ remains within $\mathcal{C}$ for all future times. To illustrate the concept, Figure 2-3 is constructed, depicting a safe set and an unsafe set. For the former set, forward invariance holds because the system trajectory remains within the circular set. The latter, unsafe set, clearly shows the trajectories leaving the set, violating the forward invariance principle [8, 10].

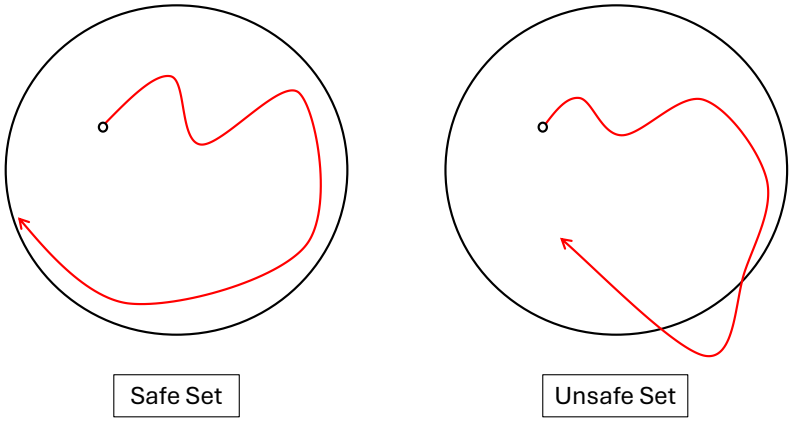



**Figure 2-3:** Illustration of a Safe Set and an Unsafe Set.

To enforce forward invariance, barrier functions leverage Nagumo's theorem [22, 8]. More precisely, consider the autonomous system

$$\dot{x} = f(x), \tag{2-19}$$

where $x \in \mathcal{X} \subseteq \mathbb{R}^n$ is the state of the system, and $f : \mathcal{X} \to \mathcal{X}$ represents its continuous-time dynamics. Now, the safe set $C$ can be defined as a superlevel set of a continuously differentiable function $h : \mathcal{X} \to \mathbb{R}$, as follows:

$$C = \{x \in \mathcal{X} \mid h(x) \geq 0\}. \tag{2-20}$$

Here, $h(\cdot)$ is a function that encodes safety as a constraint to be satisfied along state trajectories, where $h(x) > 0$ is a point on the interior of the safe set and $h(x) = 0$ on the boundary of the safe set $\partial C$. In Figure 2-3, the black outline marks $\partial C$.

To ensure forward invaraince, Nagumo's theorem states that, for the set $C$ to be invariant, the derivative of $h(x)$ must be non-negative on the boundary of $C$:

$$C \text{ is invariant} \quad \iff \quad \dot{h}(x) \geq 0, \quad \forall x \in \partial C. \tag{2-21}$$

Under this condition, trajectories that reach the boundary are either steered along the boundary $\partial C$ when $\dot{h}(x) = 0$, or back into $C$ when $\dot{h}(x) > 0$. As such, the condition guarantees that once the systems start within the safe set $C$ it never leaves it, avoiding any unsafe behavior and enforcing forward invariance.

### 2-3-2 Control Lyapunov Functions

Lyapunov functions and CLFs play a central role in the stability analysis of nonlinear systems [19]. Lyapunov functions are scalar certificates of stability and under strict decrease conditions they also certify convergence to the equilibrium. Specifically, a Lyapunov function $V : \mathcal{X} \to \mathbb{R}$ is a positive definite function that decreases along system trajectories [19]. The Lyapunov function must satisfy the following conditions:

$$V(0) = 0, \qquad V(x) > 0, \ \forall x \neq 0, \qquad \dot{V}(x) = \nabla V^\top(x)\dot{x} \leq 0, \ \forall x \neq 0. \tag{2-22}$$

The non-positivity condition of $\dot{V}(x)$ implies that $V(x)$ is non-increasing, so every closed sublevel set of $V(x)$ is forward invariant. When the Lyapunov function is strict (i.e., $\dot{V}(x) < 0$ for $x \neq 0$), the trajectory then converges to the equilibrium [19].

Building on the concept of Lyapunov functions, CLFs were introduced to extend stability guarantees to control systems [29, 19]. The idea is to select control actions that can still enforce the Lyapunov decrease condition and thus preserve stability. More precisely, a CLF $V : \mathcal{X} \to \mathbb{R}$ ensures that there exists a control input $u \in \mathcal{U} \subseteq \mathbb{R}^m$ such that the derivative of $V(x)$ can be made negative:

$$\inf_{u \in \mathcal{U}} \dot{V}(x, u) \leq 0, \quad \forall x \neq 0, \tag{2-23}$$

where $\mathcal{U}$ is the set of feasible control actions. In other words, for every state $x \neq 0$, if a valid CLF that satisfies equation (2-22) is found, it can be leveraged to ensure system stability [29, 19]. Additionally, a CLF with a decay-rate guarantee can also be imposed:

$$\inf_{u \in U} \dot{V}(x, u) \leq -\gamma_{\text{CLF}}(V(x)), \quad \forall x \neq 0, \tag{2-24}$$

where $\gamma_{\text{CLF}}(\cdot)$ is a class $\mathcal{K}^1$ function.

While CLFs are effective in guaranteeing stability, they do not inherently handle safety constraints. The introduction of barrier Lyapunov functions is an attempt to address this gap by providing invariant level sets that ensure safety. However, the barrier Lyapunov functions are often overly conservative as they require invariance for every level set of the function. This requirement limits the flexibility of the control strategy [8]. Such conservatism motivated the development of CBFs, which focus specifically on maintaining safety in a less restrictive manner by requiring the invariance only over the safety set [8].

### 2-3-3 Control Barrier Functions

CBFs provide a safety-oriented counterpart to CLFs by prioritizing safety over stability. In contrast to CLFs, which aim to drive the system state toward a stable region, CBFs are designed to ensure that the system state remains within a predefined safe set $C$ [8]. Consider the continuous-time control-affine system

$$\dot{x} = f(x) + g(x)u, \tag{2-25}$$

[1] Class $\mathcal{K}$ definition: A continuous function $\alpha : [0, a) \to [0, \infty)$ is said to belong to class $\mathcal{K}$ if it is strictly increasing and $\alpha(0) = 0$. [19]

where $f : \mathcal{X} \to \mathcal{X}$ and $g : \mathcal{X} \to \mathbb{R}^{n \times m}$ are locally Lipschitz. The safe set $C$ for this system is defined as the superlevel set of a continuously differentiable function $h : \mathcal{X} \to \mathbb{R}$:

$$C = \{x \in \mathcal{X} \mid h(x) \geq 0\}. \tag{2-26}$$

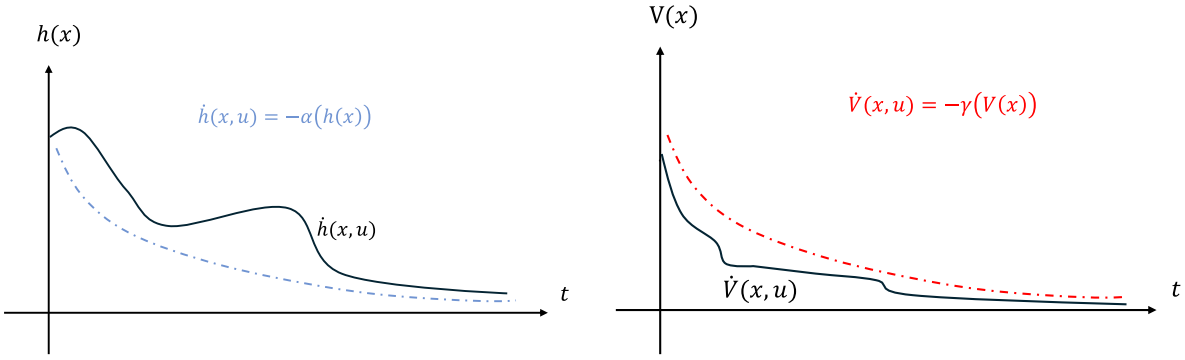
The goal of a CBF is to ensure that the system trajectories do not leave this safe set $C$. To achieve this, CBFs impose constraints on how $h(x)$ changes over time as the system evolves. Specifically, CBFs regulate the time derivative $\dot{h}(x, u)$ across the safe set so trajectories are not driven out, and at the boundary the condition forbids outward crossing. The CBF condition for the continuous-time system is defined as:

$$\sup_{u \in U} \dot{h}(x, u) = \sup_{u \in U} \left[\nabla h(x)^\top (f(x) + g(x)u)\right] \geq -\alpha(h(x)), \quad \forall x \in C, \tag{2-27}$$

where $\alpha$ is a class $\mathcal{K}$ function. The use of a class $\mathcal{K}$ function introduces flexibility in how strictly the safety constraints is enforced [10, 8]. To understand it visually, see Figure 2-4a, where the blue dotted curve represents the boundary of the CBF condition $\dot{h}(x, u) = -\alpha(h(x))$ and the black curve represents the trajectory of $\dot{h}(x, u)$. In other words, the blue curve acts as a lower bound on $\dot{h}(x, u)$, where the CBF condition ensures that $\dot{h}(x, u)$ does not drop below this bound. Far from the boundary, the CBF condition permits larger negative values of $\dot{h}(x, u)$, whereas near the boundary, the permissible decrease in of $\dot{h}(x, u)$ approaches zero. As such, when the CBF condition is satisfied, there exists a control input that keeps the trajectories within the safe set $C$, thereby preserving safety at all times [8].

CBFs are particularly effective in systems where safety is critical but stability to a specific equilibrium is not necessarily the primary objective. By ensuring forward invariance of the safe set, CBFs guarantee that the system operates within safe bounds, as such safety is guaranteed regardless of whether the system reaches a specific stable equilibrium state [10, 8].

To further highlight the difference between CLF and CBF configurations, see Figure 2-4. For the CLF condition, in Figure 2-4b, the CLF imposes an upper bound on $\dot{V}(x, u)$ enforcing a monotonic decrease of $V(x)$ and driving the state towards a target equilibrium or trajectory. In contrast, Figure 2-4a illustrates the CBF lower bound condition on $\dot{h}(x, u)$ previously discussed, which keeps the system within this safe set and prevents outward crossing.




**(a)** CBF condition visualized **(b)** CLF conditon visualized

**Figure 2-4:** CBF and CLF conditions visualized. (a): enforcing $\dot{h}(x, u) \geq -\alpha(x)$ keeps the state within the safe set (safety). (b): enforcing $\dot{V}(x, u) \leq -\gamma(x)$ drives the system toward the target (stability).

The concept outlined above holds only for continuous-time systems. The previously defined concept of CBFs can be extended to the discrete-time domain [1]. Now, consider a discrete-time control system of the form

$$x_{k+1} = f(x_k, u_k), \tag{2-28}$$

where $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is a function representing the discrete-time dynamics. The safe set $C$ is again defined as the superlevel set of a function $h(\cdot)$:

$$C = \{x_k \in \mathcal{X} \mid h(x_k) \geq 0\}. \tag{2-29}$$

To ensure that the state remains within the safe set $C$, the discrete CBF condition is given by

$$h(x_{k+1}) - h(x_k) \geq -\alpha(h(x_k)) \quad \forall x_k \in C, \tag{2-30}$$

where the set $C$ is rendered control invariant as long as there exists a control input $u_k$ that satisfies this inequality. In this condition, $\alpha$ is a class $\mathcal{K}$ function that satisfies $\alpha(r) < r$ for all $r > 0$. This condition limits the rate at which $h(x_k)$ may decrease by lower bounding the forward difference $h(x_{k+1}) - h(x_k)$ rather than the time derivative $\dot{h}(x, u)$. It avoids computing $\dot{h}(x, u)$ and is useful in discrete-time settings, for example when embedding CBFs in MPC, as will be shown further on. Similarly to the continuous-time case, the class $\mathcal{K}$ function $\alpha$ provides flexibility in enforcing the safety condition [1].

The CBFs explained and studied in this thesis belong to the category of Zeroing CBFs (ZCBF) [10]. The main idea of a Zeroing CBF, is that as the system trajectory approaches the boundary of this safe set, the value of the barrier function approaches zero, hence the name 'zeroing' function [10]. Moreover, it should be noted that the different types of CBFs covered in the following sections would be an extension of the ZCBF. However, other types of CBFs do exist like the Reciprocal CBF, but they will not be handeled in this thesis.

### Relative Degree

An important concept within CBFs is relative degree. It is defined as the number of times you need to differentiate the safety condition before the control input explicitly appears in the expression. For example, consider a simple system with position $x$, velocity $v$, and control input $u$ acting as acceleration:

$$\dot{x} = v, \quad \dot{v} = u. \tag{2-31}$$

If the safety condition is defined on the position, for example $h(x) = x - x_{\min} \geq 0$, then differentiating once gives $\dot{h}(x) = v$, which does not yet contain $u$. Differentiating a second time yields $\ddot{h}(x) = u$, where the control input finally appears. Thus, this constraint has relative degree 2. On the other hand, if the safety condition is defined on velocity, for example $h(v) = v_{\max} - v \geq 0$, then differentiating once immediately gives $\dot{h}(v) = -u$, so the control input appears after just one derivative, meaning its of relative degree one.

The proposed methods in this thesis do not employ CBFs with relative degree higher than one. However, CBFs with higher relative degree are presented here, as the concept is fundamental to

the broader CBF framework and necessary for understanding one of the approaches discussed in the literature review. To formally present the concept, consider the following control-affine system

$$\dot{x} = f(x) + g(x)u, \quad y = h(x), \tag{2-32}$$

where $y$ is the output set equal to $h(x)$. The relative degree of $h(x)$, with respect to the given dynamics in 2-32, is the number of times $h(x)$ must be differentiated with respect to time until the control input $u$ explicitly appears [24, 8]. Differentiating once gives

$$\dot{h}(x) = \frac{\partial h}{\partial x}\dot{x} = \frac{\partial h}{\partial x}f(x) + \frac{\partial h}{\partial x}g(x)u = L_f h(x) + L_g h(x)u, \tag{2-33}$$

where $L_f$ and $L_g$ denote the Lie derivatives[2]. If $L_g h(x) \neq 0$, the input appears in the first derivative and $h(x)$ has relative degree 1. If $L_g h(x) = 0$, further differentiation is needed. A second differentiation yields

$$\ddot{h}(x) = L_f^2 h(x) + L_g L_f h(x)u. \tag{2-34}$$

So if $L_g L_f h(x) \neq 0$, the relative degree is 2. More generally, the relative degree of $h(x)$ is the smallest $r$ such that

$$h^{(r)}(x) = L_f^r h(x) + L_g L_f^{r-1} h(x)u, \tag{2-35}$$

where $L_g L_f^{r-1} h(x) \neq 0$ ensures the input $u$ explicitly appears. The barrier function $h(x)$ is then said to have relative degree $r$ with respect to the system dynamics [24, 8, 33].

This motivates the Higher Order CBF (HOCBF) formulation, which extends standard CBFs to systems with relative degree $r > 1$. HOCBFs achieve this by constructing a hierarchy of auxiliary functions, where each level incorporates a class $\mathcal{K}$ function that enforces a stabilizing condition on the previous derivative. In this way, HOCBFs can be interpreted as a recursive chain of barrier-like conditions applied to successive derivatives of $h(x)$, ensuring that safety is preserved until the control input explicitly enters the dynamics.

To define an HOCBF, consider a sequence of auxiliary functions $\psi_i(x)$ for $i \in \{0, \dots, r\}$:

$$\psi_0(x) = h(x), \tag{2-36}$$

$$\psi_1(x) = \dot{\psi}_0(x) + \alpha_0(\psi_0(x)), \tag{2-37}$$

$$\vdots$$

$$\psi_i(x) = \dot{\psi}_{i-1}(x) + \alpha_{i-1}(\psi_{i-1}(x)), \tag{2-38}$$

where each $\alpha_i$ is a class $\mathcal{K}$ function that shapes how aggressively the safety constraint is enforced at that level.

The HOCBF condition is then given by

$$L_f^r h(x) + L_g L_f^{r-1} h(x)u + \alpha_r(\psi_{r-1}(x)) \geq 0. \tag{2-39}$$

The recursive structure ensures that the auxiliary functions $\psi_i(x)$ remain nonnegative, and together with the class $\mathcal{K}$ functions, this guarantees that the safe set $C$ is rendered control invariant. Thus, even when the control input $u$ influences $h(x)$ only after multiple differentiations, the system remains within the safe set [33]. An equivalent HOCBF framework for discrete-time systems is presented in [35].

[2]Lie derivatives are defined as $L_f h(x) = \frac{\partial h}{\partial x} f(x)$ and $L_g h(x) = \frac{\partial h}{\partial x} g(x)$, which represent the rate of change of $h(x)$ along the vector fields $f(x)$ and $g(x)$, respectively.

### 2-3-4 CBF-based Controllers

Having covered the fundamentals of CBFs, their main use cases in control systems will be explained in this section. CBFs are typically employed in optimization-based controllers. In literature, this is most commonly seen in the CLF-CBF-QP (Control Lyapunov Function - Control Barrier Function Quadratic Program) formulation and the MPC-CBF (Model Predictive Control with Control Barrier Functions) framework discussed below.

#### CLF-CBF-QP Formulation

The continuous-time CLF-CBF-QP formulation combines stability and safety constraints into a single optimization problem [8, 10]. The control input $u(x)$ is obtained by minimizing a quadratic cost function:

$$\begin{aligned} \min_{u \in \mathcal{U}, \sigma} \quad & \tfrac{1}{2}\, u^\top H(x) u + p\sigma^2 && \text{(2-40a)} \\ \text{s.t.} \quad & L_f V(x) + L_g V(x)\, u \leq -\gamma_{\text{CLF}}(V(x)) + \sigma, && \text{(2-40b)} \\ & L_f h(x) + L_g h(x)\, u \geq -\alpha(h(x))\,, && \text{(2-40c)} \end{aligned}$$

where $L_f$ and $L_g$ denote the Lie derivatives.

In the CLF-CBF-QP above, $H(x) \succ 0$ weights the control effort and $\sigma$ is a slack on the CLF constraint (2-40b) to ensure the condition can be relaxed when safety needs to be guaranteed. The variable $p\sigma^2$ penalizes the use of slack variable $\sigma$, so it is used only when necessary. Constraint (2-40c) is the CBF condition enforcing safety safety [8, 10]. An equivalent discrete-time version is given in [38].

#### MPC-CBF Formulation

The MPC-CBF formulation integrates the optimal control capabilities of MPC with the safety guarantees provided by the CBF formulation. The combination ensures that the system maintains both optimal performance and safety by encoding thr CBF as a constraint within the MPC framework [38]. Therefore, the optimization problem can be formulated as follows:

$$\begin{aligned} \min_{X, U} \quad & \ell_f(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k) && \text{(2-41a)} \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k), \quad k = 0, \ldots, N-1, && \text{(2-41b)} \\ & x_k \in \mathcal{X}, \quad k = 0, \ldots, N, && \text{(2-41c)} \\ & u_k \in \mathcal{U}, \quad k = 0, \ldots, N-1, && \text{(2-41d)} \\ & x_0 = x_{\text{current}}, && \text{(2-41e)} \\ & x_N \in \mathcal{X}_f, && \text{(2-41f)} \\ & h(x_{k+1}) - h(x_k) \geq -\alpha\big(h(x_k)\big), \quad k = 0, \ldots, N-1. && \text{(2-41g)} \end{aligned}$$

The discrete-time CBF constraint, in equation (2-41g), ensures that the predicted state at each time step remains in a safe set over the entire prediction horizon [38].

Since the MPC-CBF scheme predicts the evolution of the dynamics along the horizon, it is inherently less myopic than the CLF-CBF-QP, which only considers the next time step. This allows the MPC to possibly find both safer and more performing trajectories. Additionally, because MPC-CBF enforces the safety condition over a receding horizon, it can improve feasibility by anticipating future violations and selecting trajectories that remain safe, unlike CLF-CBF-QP which enforces the CBF only at the next step. However, the MPC-CBF formulation often has higher computational complexity than the CLF-CBF-QP due to solving a receding horizon optimization problem [38].

#### Input Constraints

Feasibility issues often arise in CBF-based controllers when input constraints are present. These constraints are typically actuator limits and can be represented as $\mathcal{U} = \{u \mid u_{\min} \leq u \leq u_{\max}\}$. Such bounds reduce the admissible control inputs so that, for some states, no $u \in \mathcal{U}$ satisfies the CBF inequality, making the problem infeasible. Equivalently, input constraints shrink the forward invariant safe set by excluding actions that would otherwise maintain invariance.

### 2-3-5 Exponential Control Barrier Functions

Exponential Control Barrier Functions (eCBFs) extend CBFs by selecting a linear class $\mathcal{K}$ function, $\alpha(h) = k_b h$, which imposes an exponential-decay bound in the safety constraint [24, 8]. The scalar gain $k_b$ acts as a tunable parameter that determines how fast trajectories may approach the boundary of the safe set. For a control-affine system, as defined in (2-25), with a safe set $C = \{x \in \mathcal{X} \mid h(x) \geq 0\}$, the eCBF condition is given by

$$\sup_{u \in \mathcal{U}} \left(L_f h(x) + L_g h(x) u + k_b h(x)\right) \geq 0, \tag{2-42}$$

where $k_b > 0$. Equivalently, the condition can also be written in the following exponential-decay form:

$$h(x_t) \geq h(x_0) e^{-k_b t}, \tag{2-43}$$

where $h(x_0)$ is the initial CBF value evaluated at the initial state $x_0$ [24, 8].

When extending the framework to discrete-time systems, similar principles apply. For discrete-time systems of the form (2-28) with a safe set, $C = \{x_k \in \mathcal{X} \mid h(x_k) \geq 0\}$, the eCBF counterpart is

$$h(x_{k+1}) - h(x_k) \geq -\gamma h(x_k), \tag{2-44}$$

where $\gamma \in (0, 1]$ is a parameter governing the exponential decay in the discrete-time setting. This condition ensures that at each time step, the barrier function decreases by a fixed proportion of its current value. This proportional decrease enforces an exponential decay over in discrete time, as given by the equation below:

$$h(x_k) \geq h(x_0)(1 - \gamma)^k. \tag{2-45}$$

This discrete-time condition mirrors the exponential decay discussed in the continuous-time case and ensures that the system state remains within the safe set while decaying at a controlled rate over time [1].

The discrete eCBF condition is central to several methods in this thesis and the parameter $\gamma$ (the decay rate) plays a key role. To build intuition about its influence, Figure 2-5 shows trajectories for several $\gamma$ values in a simple obstacle-avoidance task with a circular obstacle. A larger decay rate $\gamma$ makes the per-step lower bound $(1-\gamma)h(x_k)$ decay more rapidly. As a result, a smaller $h(x_{k+1})$ is admissible at the next time step, leading to less conservative behavior because the solver can choose values closer to the safety boundary. This can be seen in Figure 2-5, where with $\gamma = 0.9$ the trajectory runs along the boundary of the safe set. A smaller decay rate $\gamma$ makes the per-step lower bound $(1-\gamma)h(x_k)$ decay more slowly. As a result, $h(x_{k+1})$ cannot drop as low as it could with a larger $\gamma$, leading to more conservative behavior because the solver must choose values farther from the safety boundary. This can be seen in Figure 2-5, where with $\gamma = 0.1$ the trajectory stays well inside the safe set, far from the boundary.

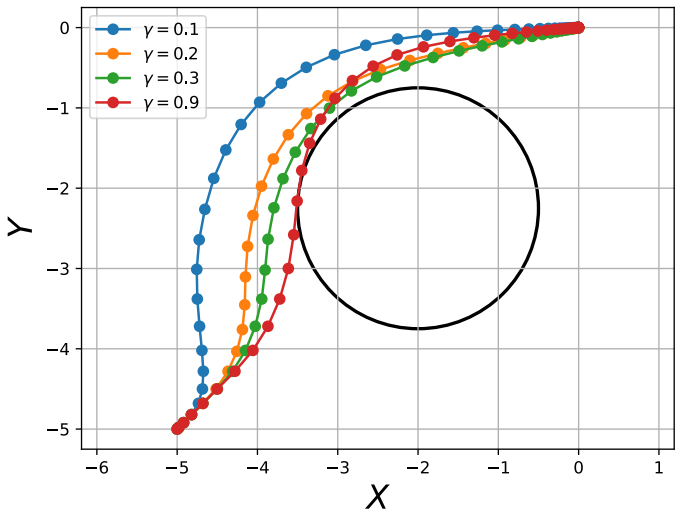



**Figure 2-5:** Different $\gamma$ (decay rate) values for an MPC-CBF with D-eCBF and horizon N=8.

### 2-3-6 Optimal Decay CBFs

As discussed earlier, a major limitation of nominal CBFs is their inability to handle infeasibilities caused by input constraints. Optimal Decay Control Barrier Functions (OPTD-CBFs) address this by optimizing the decay rate $\gamma$ (or $k_b$ in the continuous case) of the eCBF [37]. OPTD-CBF replaces the fixed decay rate with a dynamic decay variable $\omega$ leading to

$$L_f h(x) + L_g h(x) \geq -\omega h(x). \tag{2-46}$$

The decay term is then optimized within the OPTD-CBF-QP by treating $\omega$ as a descion variable:

$$\min_{u \in \mathcal{U}, \omega \in (0,1]} \quad \tfrac{1}{2}\|u - k(x)\|^2 + P_\omega(\omega - \bar{\omega})^2 \tag{2-47a}$$

$$\text{s.t.} \quad L_f h(x) + L_g h(x) u \geq -\omega(h(x)), \tag{2-47b}$$

$$u \in \mathcal{U}. \tag{2-47c}$$

where $P_\omega$ and $\bar{\omega}$ are scalars used to tune performance [37] and $k : \mathcal{X} \to \mathcal{U}$ is a nominal controller. Here $\bar{\omega}$ serves as a reference decay rate that guides the value of $\omega$. The weight $P_\omega$ regulates how tightly $\omega$ is pulled towards $\bar{\omega}$. A smaller $P_\omega$ reduces the penalty for deviations of $\omega$ from $\bar{\omega}$, making $\omega$ more adaptive. Conversely, a larger $P_\omega$ keeps $\omega$ closer to $\bar{\omega}$. Moreover, as $P_\omega \to \infty$, $\omega$ is forced to always equal $\bar{\omega}$, whereby the formulation becomes equivalent to the nominal exponential CBF constraint with decay rate given by $\bar{\omega}$ [37]. The choice of $P_\omega$ and $\bar{\omega}$ are design decisions made by the the practitioner to balance adaptivity of $\omega$.

Additionally, the concept of optimal decay was extended to the nonlinear MPC (NMPC) formulation in [36]. In that work, the linear decay rate $\gamma_k$ in the exponential CBF is held fixed rather than treated as a decision variable, as in the QP above (2-47). Instead, the NMPC formulation introduces additional decision variables $\Omega = [\omega_0, \ldots, \omega_{M_{\text{CLF}}-1}]^T$ to relax the CBF constraint through $h(x_{k+1}) \geq \omega_k (1 - \gamma_k) h(x_k)$. This relaxation enhances both performance and feasibility by allowing the controller to adapt the safety margin dynamically across the prediction horizon. To regulate the $\Omega$ variables, the objective includes $\psi(\omega_k)$,which penalizes deviations of $\omega_k$ from $\bar{\omega}$ via $P_\omega(\omega - \bar{\omega})^2$. This follows the same explanation as in (2-47). Furthermore, the NMPC problem also includes slack variables $\Sigma = [\sigma_0, \ldots, \sigma_{M_{\text{CLF}}-1}]^T$ to relax the CLF stability conditions when necessary. The CLF constraint includes a linear decay rate $\rho_k > 0$, which serves as the linear class $\mathcal{K}$ function in the CLF condition, similar to the decay rate in the eCBF. The full NMPC is then given as follows [36]:

$$J(x_k) = \min_{X,\, U,\, \Omega,\, \Sigma} \quad \ell_f(x_N) + \sum_{k=0}^{N-1} \ell(x_{k+1},\, u_k) \;+\; \psi(\omega_k) \;+\; \phi(\sigma_k) \tag{2-48a}$$

$$\text{s.t.} \quad x_{k+1} = f(x_k,\, u_k), \qquad k = 0, \ldots, N-1, \tag{2-48b}$$

$$u_k \in \mathcal{U}, \quad x_k \in \mathcal{X}, \qquad k = 0, \ldots, N-1, \tag{2-48c}$$

$$x_0 = x_{\text{current}}, \tag{2-48d}$$

$$h(x_{k+1}) \;\geq\; \omega_k \,(1 - \gamma_k)\, h(x_k), \qquad k = 0, \ldots, M_{\text{CBF}} - 1, \tag{2-48e}$$

$$\omega_k \geq 0, \qquad k = 0, \ldots, M_{\text{CBF}} - 1, \tag{2-48f}$$

$$V(x_{k+1}) \leq (1 - \rho_k)\, V(x_k) + \sigma_k, \qquad k = 0, \ldots, M_{\text{CLF}} - 1. \tag{2-48g}$$

Here, $M_{\text{CBF}}$ and $M_{\text{CLF}}$ denote the horizon lengths over which the CBF and CLF constraints are enforced, respectively. These horizons can be set equal to the prediction horizon $N$ or chosen smaller, depending on whether the constraints should be enforced over the entire horizon. The objective also includes $\phi(\sigma_k) \geq 0$ to penalize the CLF slack variables $\Sigma$ [36].

# Chapter 3

# Literature Review

With the rise of machine learning, data-driven techniques have enabled the direct synthesis of CBFs or the adjustment of the CBF condition. Instead of relying solely on explicit models, these methods use data collected through interactions with a dynamical system. In this chapter, a literature review is conducted to cover two types of learning-based CBF methods. The first and the one most relevant to the methods developed in this thesis focuses on using data-driven approaches to amend the CBF condition for a given and fixed CBF $h(x)$. These methods learn to adjust the CBF condition either by learning the class $\mathcal{K}$ function itself or by introducing a learnable penalty term. For example, the method presented in the introduction of this thesis by Sabouni et al. [26] belongs to this category and will be discussed in detail in this section. The second type briefly considers approaches that learn the CBF $h(x)$ itself from data. While this lies outside the scope of this thesis, it is included here to provide perspective on a different way through which data-driven techniques can contribute to safety.

## 3-1 Data-Driven Shaping of CBF Constraints

### 3-1-1 Adaptive class $\mathcal{K}$ in RL-MPC with CBF

Recall the Sabouni method introduced in Chapter 1. This subsection examines this method in greater detail. As outlined earlier, the approach combines MPC–CBF with deep RL to learn a parameterized MPC together with parameterized CBF and CLF constraints that feed into the MPC [26]. The CBF condition is formulated in continuous time and enforced at

discrete-time steps within the MPC. The resulting MPC–CBF problem is

$$\begin{aligned}
\min_{X,U,\sigma} \quad & \ell_{f,\theta_{o,k}}(x_N) + \sum_{k=0}^{N-1} \ell_{\theta_{o,k}}(x_k,\, u_k) + \theta_{e,k}^T \sigma^2 && \text{(3-1a)}\\
\text{s.t.} \quad & x_{k+1} = f(x_k,\, u_k), \qquad k = 0, \ldots, N-1, && \text{(3-1b)}\\
& u_k \in \mathcal{U}, \qquad k = 0, \ldots, N-1, && \text{(3-1c)}\\
& x_k \in \mathcal{X}, \qquad k = 0, \ldots, N, && \text{(3-1d)}\\
& x_0 = x_{\text{current}}, && \text{(3-1e)}\\
& L_f V(x) + L_g V(x) u \leq -\gamma_{\theta_c}(V(x)) + \sigma, && \text{(3-1f)}\\
& L_f h(x) + L_g h(x) u \geq -\alpha_{\theta_c}(h(x)). && \text{(3-1g)}
\end{aligned}$$

The learnable parameters are grouped as $\theta_k = [\,\theta_{o,k},\ \theta_{c,k},\ \theta_{e,k}\,]^\top$ where $\theta_{o,k}$ parameterizes the MPC stage and terminal costs, $\theta_{c,k}$ parameterizes the class-$\mathcal{K}$ functions in the CBF/CLF constraints and $\theta_{e,k} \geq 0$ is a parametrized penalty weight on the CLF slack $\sigma$ to keep the CLF constraint soft. The class $\mathcal{K}$ function can be parameterized by any standard class $\mathcal{K}$ function, as long as the class $\mathcal{K}$ properties are satisfied. These two properties are that it is strictly increasing and equals zero at the origin. In [26], a linear class $\mathcal{K}$ function is used, $\alpha_{\theta_{c,k}}(y) = \theta_{c,k} y$ with $\theta_{c,k} \in \mathbb{R}_{>0}$. The same parametrization also extends naturally to HOCBFs, with each class $\mathcal{K}$ function corresponding to the higher-order derivatives of $h(x)$ parameterized in the same way (see Section 2-3-3 for a review of relative degree and HOCBFs).

The learning setup in this paper differs from the MPC-based RL method described earlier in Section 2-2-1. In this method, the RL state is $s_k = [x_k, u_k]^T$ and the RL action is the parameter vector $\theta_k$. This allows a policy $\pi_w : \mathbb{R}^{n+m} \to \mathbb{R}^{n_\theta}$ to map $s_k$ to the parameter vector $\theta_k$, whereby, upon deployment, the MPC samples this parameter vector from the policy at each time step using the environment state $x_k$ and the current MPC action $u_k$, as shown in Figure 3-1. Training follows a deep actor–critic design that uses NNs for both policy and value function approximations. The policy $\pi_w$ is a network with weights $w$, while the critic $Q_v$ is a value function network with weights $v$. The full pipeline is summarized in Figure 3-1 [26].

Since the policy outputs a new parameter vector $\theta_k$ at each time step, the class-$\mathcal{K}$ function becomes time-varying and enables more expressive safety relaxations than a fixed form such as $\alpha(y) = \gamma y$. An additional benefit of this method is lower computational cost compared to MPC-based RL because it avoids backpropagation through the MPC solver. By contrast, MPC-based RL can be more sample efficient. Differentiating through the MPC yields richer gradients updates for learning, which can reduce the number of RL iterations. This advantage becomes even more substantial when second-order updates are used. Lastly, because the safety parameters are only updated across episodes and remain fixed within each rollout, there is no guarantee of safety or feasibility during training.

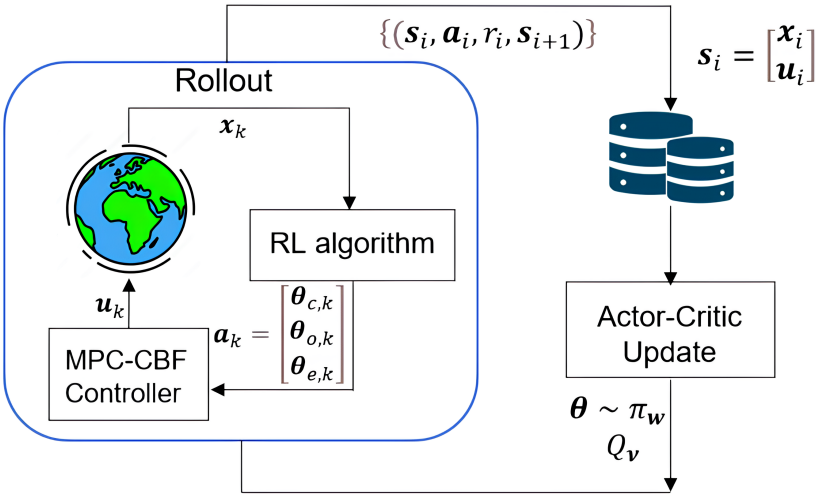



**Figure 3-1:** RL training pipeline for the MPC-CBF in [26].

### 3-1-2 Learning class $\mathcal{K}$ for CBF

The class-$\mathcal{K}$ function in the CBF condition is often manually designed, for example as linear or quadratic functions, which limits its expressiveness and adaptability. As a result, it cannot easily capture the range of system dynamics or scenario-specific requirements. To address this limitation, in [13], Chriat and Sun propose an Adaptive Multi-step Control Barrier Function (AM-CBF). In their method, the class-$\mathcal{K}$ function is parameterized by a neural network and trained jointly with a reinforcement learning policy. The training is performed over multiple rolled out QP steps to mitigate the short-sighted behavior of traditional CBF-QPs (see Section 2-3-4), while execution remains efficient through a single-step CBF-QP that projects the policy action into the set of safe control actions. Gradients are propagated through this QP, which enables the entire pipeline to be trained end-to-end [13].

To ensure the learned network defines a valid class $\mathcal{K}$ function, the paper enforces its required properties. Here $\kappa(\cdot)$ denotes the class $\mathcal{K}$ function. It must satisfy two properties: it is monotonically increasing and it is zero at zero, i.e., $\kappa(0) = 0$. Monotonicity is enforced by constraining the network weights (excluding biases) to be non-negative, achieved by parameterizing the weights using absolute-value or exponential functions, which ensures that the network is monotonically increasing. The zero-at-zero condition is enforced by shifting the neural network output according to $\kappa(z) = \kappa'(z) - \kappa'(0)$. Satisfying both conditions yields a learned class $\mathcal{K}$ function and the architecture is shown in Fig. 2(b) [13].

The class $\mathcal{K}$ function $\kappa(\cdot)$ is incorporated into the CBF condition, which is subsequently embedded in a QP formulation. The purpose of this formulation is to project the nominal action from the RL policy, $u_{\text{RL}}$, to a safe action $u_r$:

$$\arg\min_{u_r} \quad \|u_r - u_{\mathrm{RL}}\|^2 \tag{3-2a}$$

$$\text{s.t.} \quad u_{min} \leq u_r \leq u_{max}, \tag{3-2b}$$

$$\frac{\partial h}{\partial x}\Big(f(x) + g(x)u_r\Big) \geq -\kappa(h(x)). \tag{3-2c}$$

where $u_{min}$ and $u_{max}$ denote the input bounds. The policy of the RL algorithm is approximated by a deep neural network. The RL policy and the QP with the neural $\kappa(\cdot)$ are trained together by differentiating through the QP. More precisely, differentiation is done through using the KKT conditions, which due to convexity of the QP can guarantee global optimality [13]. This is analogous to computing sensitivities in MPC-based RL formulations, where they are obtained by differentiating the Lagrangian at the primal-dual solution, as described in Section 2-2-3. As a result, both the policy network and the class $\mathcal{K}$ network are updated simultaneously to maximize an expected reward (or minimize an expected cost). To reduce myopic behavior, obtain more optimal trajectories, and avoid infeasibility caused by input constraints, the RL agent collects rewards over multiple steps within each rollout, as shown in Figure 3-2. At deployment only one QP is solved per step, which keeps execution efficient. This stands in contrast to the method of [26], which relies on repeatedly solving an MPC [13].

Once again, because safety is enforced only one time step ahead and the safety parameters remain fixed within each rollout, the controller is more prone to infeasibility during exploration in training, similar to what was noted for [26]. However, this issue is more severe here since the QP enforces safety only one step ahead, whereas the MPC formulation in [26] provides some foresight to preserve feasibility.

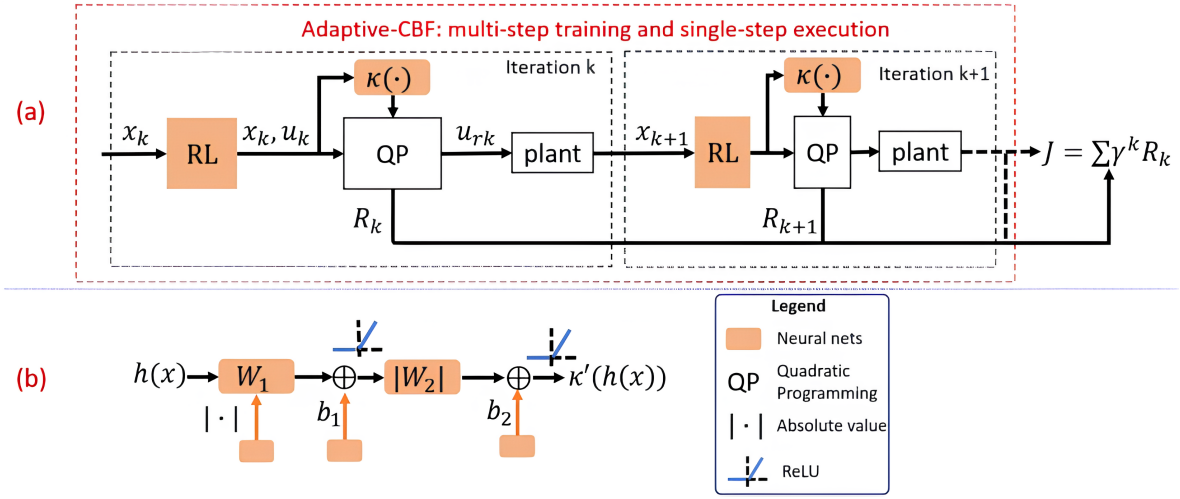



**Figure 3-2:** Neural network architecture for learning the class $\mathcal{K}$ function in AM-CBF [13].

### 3-1-3 BarrierNet: Learned Penalty Terms in CBF Constraints

Similar to the previous method [13], the approach in [34] integrates safety constraints into an end-to-end trainable architecture. Instead of directly learning the class $\mathcal{K}$ function, [34] intro-

duces a penalty function $p(z)$ that modulates the CBF conditions as $p(z)\alpha(h(x))$, providing a more flexible and adaptive safety layer within the policy network.

The key idea is that multiplying the class $\mathcal{K}$ function by a positive penalty function $p(z)$ still preserves the CBF condition. As long as $p(z) > 0$ and it is Lipchitz continuous, the resulting $p(z)\alpha(h(x)))$ remains a valid class $\mathcal{K}$ function. The penalty function as a result can be amended to reduce the conservativeness typically caused by fixed $\mathcal{K}$ terms [34].

However, the choice of the penalty function remains arbitrary. To address this, the paper proposes parameterizing $p(z)$ with a neural network to learn a flexible penalty function. The inputs $z$ correspond to neural network inputs, which are a design choice and may include environmental features such as obstacle positions or velocities. In addition, the approach is combined with HOCBFs (see Section 2-3-3), where the neural network outputs a set of penalty terms $p_i(z)$ that enter directly into the recursive HOCBF condition:

$$\psi_i(x,z) := \dot{\psi}_{i-1}(x,z) + p_i(z)\alpha_i(\psi_{i-1}(x,z)), \quad i \in \{1,\dots,m\}, \tag{3-3}$$

with $\psi_0(x,z) = h(x)$ as the barrier function defining the safe set.

The neural network is then applied in a similar way to the previous method by incorporating it into a QP to generate safe control actions, whereby the QP can be formulated as follows.

$$u^\star = \arg\min_{u} \quad \tfrac{1}{2}\, u^\top H(z \mid \theta_h)\, u + F\left(z \mid \theta_f\right)^\top u \tag{3-4a}$$

$$\text{s.t.} \quad L_f^m b_j(x) + \left[L_g L_f^{m-1} b_j(x)\right] u + O(b_j(x), z \mid \theta_p) + p_m(z \mid \theta_p^m)\, \alpha_m(\psi_{m-1}(x, z \mid \theta_p)) \geq 0, \quad j \in S, \tag{3-4b}$$

$$u_{\min} \leq u \leq u_{\max}. \tag{3-4c}$$

Here, $\theta_h$, $\theta_f$, and $\theta_p = (\theta_p^1, \dots, \theta_p^m)$ are trainable parameters that, among others, parameterize the matrices $H$, $F$, and the penalty terms $p_i(z)$. The matrices $H$, $F$ and the penalty terms $p_i(z)$ are also parameterized. The additional term $O(b_j(x), z \mid \theta_p)$ represents lower-order components from the recursive HOCBF expansion, parameterized by the network and dependent on both the state $x$ and the neural network input $z$. To guarantee Lipschitz continuity of the learned $p_i(z)$, the penalty network with parameters $\theta_p$ uses continuously differentiable activations such as Sigmoid functions. The policy network with parameters $\theta_f$ is not subject to this restriction [34].

In this setup, $F\left(z \mid \theta_f\right)$ represents the control action proposed by an external (nominal) controller, which the QP then projects onto the safe set to produce the closest control input that still satisfies the barrier constraints. The matrix $H\left(z \mid \theta_h\right)$ penalizes deviations between the nominal control action and the projected safe control action. By substituting $F = -Hu_{\text{nom}}(z)$ and completing the square, the objective function can be rewritten as

$$\frac{1}{2}(u - u_{\text{nom}}(z))^\top H(u - u_{\text{nom}}(z)) - \frac{1}{2}u_{\text{nom}}^\top(z) H u_{\text{nom}}(z). \tag{3-5}$$

As such, the key feature is that this QP-based safety filter acts as a modular safety layer that can be combined with different types of controllers, similar to the previous method. For

example, it can be attached to a neural network controller parameterized by $\theta_f$. Since the QP is differentiable, the entire combined system remains end-to-end trainable with backpropagation, which is performed using the Lagrangian and KKT conditions (see Section 2-2-3).

Similar to the previous method [13], this framework benefits from a lightweight learnable QP that projects unsafe policies from a learnable controller to a safe set of control actions. This is less computationally intensive than solving an MPC as in [26]. A strong point of this approach is that it can also handle higher relative degree CBFs, unlike [13], though [26] can address them as well. However, since only a penalty function is learned while the class $\mathcal{K}$ function remains fixed, the choice of class $\mathcal{K}$ function is still arbitrary.

Moreover, in this method, the learnable controller is trained through supervised learning by generating a set of control actions using a nominal controller and fitting to them. As such, unlike RL-based methods presented before, which can actively explore and adapt policies during training, this approach relies entirely on supervised data from a nominal controller. Its performance is therefore tied to the quality of the nominal controller used for data generation. Nevertheless, this QP framework remains differentiable and could in principle be integrated into an RL-based approach in a manner similar to [13].

## 3-2 Learning CBF From Data

### 3-2-1 Neural CBFs

Unlike traditional CBFs, where the function $h(x)$ is assumed to be explicitly known or has a simple closed-form expression, neural CBFs are employed when $h(x)$ is unknown or lacks a straightforward analytical form. In these cases, a neural network is used to approximate $h(x)$ [16]. Specifically, $h(x)$ is parametrized, and then trained using randomly sampled data points that are labeled on the basis of whether they belong to safe or unsafe sets. In [16], the training of the neural CBF is conducted using the following empirical loss function:

$$L_V = \lambda_1 \frac{1}{N_{\text{safe}}} \sum_{i=1}^{N_{\text{safe}}} \max(h(x_i), 0) + \lambda_2 \frac{1}{N_{\text{unsafe}}} \sum_{i=1}^{N_{\text{unsafe}}} \max(-h(x_i), 0) + \lambda_3 \frac{1}{N_{\text{total}}} \sum_{i=1}^{N} r(x_i). \quad (3\text{-}6)$$

In this formulation, $N_{\text{safe}}$ represents the number of samples labeled as safe, while $N_{\text{unsafe}}$ is the number of samples labeled as unsafe, and the total number of sampled states is denoted by $N_{\text{total}}$. The terms $\lambda_1$, $\lambda_2$, and $\lambda_3$ are hyper-parameters that weigh the contributions of the respective components in the loss function. The first term penalizes safe points for which $h(x) < 0$, encouraging $h(x) \geq 0$ for all points in the safe set. The second term penalizes unsafe points for which $h(x) \geq 0$, promoting $h(x) < 0$ for all points in the unsafe set. Lastly, the third term penalizes slack variables $r(x)$, which are introduced to allow small, controlled violations of the safety constraints, ensuring feasibility for control tasks [16].

This empirical loss can be minimized using stochastic gradient descent with respect to variables in the neural network. However, zero empirical loss does not guarantee a valid CBF, since the neural network is trained on a finite set of sampled points from the state space. As such, there could be an existing set of points that were not sampled and would violate the trained neural CBF. To that end, after the convergence of the training process there is a need

for verification of the learned neural CBF [16]. Verification for learned certificates such as CBFs can be achieved through probabilistic methods based on generalization error bounds, Lipschitz arguments, and optimization-based methods.

1. **Probabilistic Methods:** Probabilistic methods based on error bounds sample a subset of the state space, validate the CBF at these points, and then statistically infer its validity across the entire space. While straightforward to implement, this approach is not the most reliable as it does not provide deterministic guarantees; statistically unlikely points may still fail to satisfy the safety condition [16].
2. **Lipschitz Property-Based Methods:** An alternative method relies on the Lipschitz property, defined as $|f(x_1) - f(x_2)| \leq L \cdot \|x_1 - x_2\|$, where $L$ is the Lipschitz constant that bounds the function between two points, based on the worst-case rate of change. This property limits how much a function can change over an interval. By checking the safety constraints at sampled grid points, stability can be guaranteed between those points if the function's rate of change remains below a given threshold. However, this approach struggles with high-dimensional systems due to the curse of dimensionality and is often overly conservative, as it assumes the worst-case variation of the function [16].
3. **Optimization-Based Methods:** The optimization-based verification method runs verification alongside training. When invalid regions are detected, counterexamples are generated to refine the training process. In other words, a learner is responsible for learning the CBF itself, while a verifier periodically checks if the current CBF is valid. If the verifier determines the CBF is valid then the training stops, otherwise it feeds the learning process a counterexample to improve the learning. The verification itself is performed through optimization techniques, such as Mixed-Integer Linear Programming (MILP) or Satisfiability Modulo Theories (SMT), which systematically search the state space to identify regions where the CBF fails to satisfy the required constraints. This is effective but computationally expensive, as it requires constant interaction between the learner and verifier [16].

It is also worth noting, even though there are many other varied ways of verifying learning certificates that have not been explored in this literature review, these three are the most notable ones.

Another proposed learning-based framework for CBFs is the Policy Neural CBF (PNCBF), presented in [28]. The PNCBF constructs a CBF using a policy value function, which can be defined as $V^{h,\pi}(x_0) \coloneqq \sup_{t\geq 0} h(x_t^\pi)$, where $\pi$ is an arbitrary policy and $x_t^\pi$ is the state at time $t$ given by following the policy $\pi$ [28]. The idea is that the policy value function gives an upper bound on worst-case constraint violation on $h(x)$. The policy value function can be seen as a valid CBF since it satisfies the conditions $V^{h,\pi}(x) \geq h(x)$ and $\nabla V^{h,\pi}(x)^\top (f(x)+g(x)\pi(x)) \leq 0$ [28]. The policy value function is approximated offline using a neural network. This is done by simulating trajectories using the nominal policy ($\pi_{nom}$) to collect data points of $(x_0, \max_{0\leq t\leq T} h(x_t), x_T)$ [28]. Then, using these data points, the network is optimized through minimizing the following loss function:

$$L(\theta) = \sum \|V_\theta^{h,\pi}(x) - \max(\max_{t\leq s\leq T} h(x_s), V_\theta^{h,\pi}(x_T))\|^2. \tag{3-7}$$

Following the convergence of the training process for the approximation of $V^{h,\pi}(x_0)$, a controller based on the following QP optimization problem is set up:

$$\min_u \|u - \pi_{\text{nom}}(x)\|^2, \quad \text{s.t. } \nabla V^{h,\pi}(x)^\top (f(x) + g(x)u) \leq -\alpha(V^{h,\pi}(x)), \tag{3-8}$$

where $\nabla V^{h,\pi}(x)^\top (f(x) + g(x)u) \leq -\alpha(V^{h,\pi}(x))$ is the safety condition based on the approximated $V^{h,\pi}(x_0)$ and $\pi_{\text{nom}} : \mathbb{R}^n \to \mathbb{R}^m$ is a nominal policy [28].The QP optimization problem improves the CBF by using the policy generated by the QP optimization controller as the new nominal policy for constructing an updated PNCBF. This new nominal policy is then used to re-train the policy value function, resulting in a refined PNCBF with an enlarged safe set. The process iterates, alternating between policy improvement through the QP optimization and policy evaluation through re-training, until the safe set reaches its maximum size or no further improvement is observed [28]. This allows the PNCBF to achieve a larger safe set compared to handcrafted CBFs or methods based on HOCBFs. Moreover, since the PNCBF is also trained using sampled data and minimizes an empirical loss, it shares the same limitation as the neural CBF, as its validity is only guaranteed on the sampled points. Therefore, post-training verification of the learned PNCBF is required to ensure safety across the entire state space [28].

# Chapter 4

# Methodology

In this chapter, three novel approaches are presented to address the research question posed in Chapter 1. Each method leverages MPC-based RL, as outlined in Section 2-2-1, to tune the parameters of a parameterized MPC. These parameters include those defining the class $\mathcal{K}$ function within the discrete CBF condition of the MPC, which was introduced in Section 2-3-3 and defined in equation (2-30), to guarantee safety. In addition, the methods learn or shape the class $\mathcal{K}$ function of the discrete CBF condition, introduced in Section 2-3-3 and defined in equation (2-30), to guarantee safety. The three methods to be explored are Learnable Optimal Decay CBF (LOPTD-CBF), Neural Network CBF (NN-CBF) and Recurrent Neural Network CBF (RNN-CBF). As a general remark, these methods parameterize both the objective function and the CBF condition in the MPC. However, only the parameterization of the safety condition is discussed here, since the parameterization of the objective function for MPC-based RL has already been presented in Section 2-2-1. To avoid confusion, in the remainder of this thesis the actual system states and actions will follow the RL notation $s$ and $a$, while the predicted states and actions within the MPC will be denoted by $x$ and $u$, respectively (as introduced in Section 2-2-1). In this chapter each method's parametrized MPC is presented only as the state-value function $V_\theta(s)$ (see (2-10)), since the state–action value function $Q_\theta(s, a)$ (see (2-11)) is similar and omitted for conciseness. Within this chapter, the following two assumptions are considered:

**Assumption 1.** *The prediction model $f$ is known and coincides exactly with the true system dynamics.*

In contrast to standard MPC-based RL, the prediction model is kept fixed to preserve formal safety guarantees. Under model mismatch the CBF condition may fail. Changing the prediction model could cause violations under the true system dynamics. Learning under model mismatch requires separate treatment and lies outside the scope of this thesis.

**Assumption 2.** *The CBF is known and its safe set $\mathcal{C} = \{s \in \mathcal{S} \mid h(s) \geq 0\}$ equals the true safe set.*

Methods similair to the ones from Section 3-2-1 are not considered. RL in this chapter does not learn a CBF. The CBF is known a priori.

## 4-1 Learnable Optimal Decay CBF

**Set of Learnable Parameters:** $\{\bar{\omega}_{k,\theta}, P_{\omega_k,\theta}, \ell_\theta, \ell_{f,\theta}\}$ . The parameters $\bar{\omega}$ and $P_\omega$ are the optimal decay variables used in the OPTD-CBF framework (see Eq. (2-47) and Eq.(2-48)) to control the decay decision variable $\omega$ through the objective term $P_\omega(\omega - \bar{\omega})^2$. Here, $\bar{\omega}$ acts as a reference decay rate while $P_\omega$ is a penalty weight that regulates how strongly $\omega$ is kept near $\bar{\omega}$. Together, they determine how adaptive and conservative the decay variable $\omega$ can be when shaping the CBF constraint. In practice, a small $P_\omega$ allows $\omega$ to vary more freely. This can improve feasibility and performance. A large $P_\omega$ restricts $\omega$ to remain closer to $\bar{\omega}$, resulting in a more rigid controller. The functions $\ell_\theta$ and $\ell_{f,\theta}$ represent the stage cost and terminal cost in the parametrized MPC objective (see Section 2-1-1).

### Single CBF Constraint Formulation

A problem in OPTD CBF framework is the selection of sensible OPTD parameters $\{\bar{\omega}, P_\omega\}$. Since selecting these parameters is somewhat arbitrary and problem dependent, an RL agent is used to tune these quantities for the specific task, while safety remains enforced by the CBF constraints. Hence, building on the OPTD CBF framework in Section 2-3-6, an MPC-based RL formulation is obtained by lifting the OPTD parameters $\{\bar{\omega}, P_\omega\}$ to time-indexed, learnable parameters $\{\bar{\omega}_{k,\theta}, P_{\omega_k,\theta}\}$ over the MPC prediction horizon. The OPTD framework is adapted to use these parameters to tune the decay rate in the CBF condition at each prediction step $k$, yielding the following parametrized MPC value function $V_\theta(s)$:

$$
\begin{aligned}
V_\theta(s) = \min_{X,U,\Omega,\Sigma} \quad & \ell_{f,\theta}(x_N) + \sum_{k=0}^{N-1} \left[\ell_\theta(x_k,\, u_k) + \psi_{k,\theta}(\omega_k)\right] + w_{\text{MPC}} \sum_{k=0}^{N-1} \sigma_k && \text{(4-1a)}\\
\text{s.t.} \quad & x_{k+1} = f(x_k,\, u_k), \qquad k = 0, \ldots, N-1, && \text{(4-1b)}\\
& x_k \in \mathcal{S}, \qquad k = 0, \ldots, N, && \text{(4-1c)}\\
& x_0 = s, && \text{(4-1d)}\\
& h(x_{k+1}) - (1-\omega_k) h(x_k) \geq -\sigma_k\,, \qquad k = 0, \ldots, N-1, && \text{(4-1e)}\\
& u_k \in \mathcal{A}, \quad \omega_k \geq 0, \quad \sigma_k \geq 0, \qquad k = 0, \ldots, N-1, && \text{(4-1f)}
\end{aligned}
$$

where $\Omega = [\omega_0, \ldots, \omega_{N-1}]^T$ collects the decision variables that relax the CBF condition at each time step. The slack variables $\Sigma = [\sigma_0, \ldots, \sigma_{N-1}]^T$ maintain feasibility during training by permitting temporary CBF violations, with their use penalized in the cost through $w_{\text{MPC}}$.

Unlike the slack variables, the $\Omega$ decision variables are shaped via the adjusted OPTD penalty function $\psi_{k,\theta}(\omega_k) = P_{\omega_k,\theta}(\omega_k - \bar{\omega}_{k,\theta})^2$, where $P_{\omega_k,\theta}$ and $\bar{\omega}_{k,\theta}$ are the RL parameters defined above that govern the decay rate across the prediction horizon. This penalty function lets each $\omega_k$ be adjusted toward a reference decay rate through $\bar{\omega}_{k,\theta}$ and adjust its adaptivity through $P_{\omega_k,\theta}$. In this setting, the RL formulation benefits further, since hand-tuning these parameters across the horizon is arbitrary and difficult.

### Multiple CBF Constraints Formulation

Extending the framework to multiple CBF constraints needs to be addressed, as this situation can arise when the system must satisfy several safety conditions simultaneously. For instance, in obstacle avoidance scenarios, each obstacle would correspond to its own CBF. Therefore, extending the framework to multiple CBFs requires introducing a separate constraint for each CBF at every prediction step. Building on the previous single-CBF formulation, let $i = 1, \dots, \mathcal{O}$ denote the CBF indices. For each CBF $i$ and each prediction step $k = 0, \dots, N-1$, there is a need to introduce a decay-rate reference $\bar{\omega}_{k,i,\theta}$ a penalty weight $P_{\omega_{k,i},\theta}$ and the optimal decay decision vairables $\Omega = [\omega_{0,0}, \dots, \omega_{N-1,\mathcal{O}}]^T$. Accordingly, for each new CBF constraint, a corresponding slack variable $\sigma_{k,i}$ is added. The parametrized MPC then becomes:

$$V_\theta(s) = \min_{X, U, \Omega, \Sigma} \quad \ell_{f,\theta}(x_N) + \sum_{k=0}^{N-1} \ell_\theta(x_k,\, u_k) + \sum_{k=0}^{N-1} \sum_{i=1}^{\mathcal{O}} \psi_{k,i,\theta}(\omega_{k,i}) + w_{\text{MPC}} \sum_{k=0}^{N-1} \sum_{i=1}^{\mathcal{O}} \sigma_{k,i} \tag{4-2a}$$

$$\text{s.t.} \quad x_{k+1} = f(x_k,\, u_k), \qquad k = 0, \dots, N-1, \tag{4-2b}$$

$$x_k \in \mathcal{S}, \qquad k = 0, \dots, N, \tag{4-2c}$$

$$u_k \in \mathcal{A}, \qquad k = 0, \dots, N-1, \tag{4-2d}$$

$$x_0 = s, \tag{4-2e}$$

$$h_i(x_{k+1}) - (1 - \omega_{k,i}) h_i(x_k) \geq -\sigma_{k,i}, \qquad \begin{matrix} k = 0, \dots, N-1, \\ i = 1, \dots, \mathcal{O}, \end{matrix} \tag{4-2f}$$

$$\omega_{k,i} \;\geq\; 0, \quad \sigma_{k,i} \;\geq\; 0, \quad \begin{matrix} k = 0, \dots, N-1, \\ i = 1, \dots, \mathcal{O}. \end{matrix} \tag{4-2g}$$

However, the number of learnable parameters grows with both the MPC prediction horizon and the number of CBF constraints. Each CBF at every prediction step introduces a decay variable $\omega_{k,i}$ together with its corresponding reference $\bar{\omega}k, i, \theta$ and penalty weight $P\omega_{k,i}, \theta$. Longer horizons therefore increase not only the total number of CBF constraints and the optimal decay decision variables associated with them, but also the dimensionality of the parameter space to be tuned. Conversely, if the horizon is too short, the RL agent may lack sufficient flexibility to obtain a satisfactory solution. This strong dependence on the horizon is a principal limitation of the method. Nonetheless, the same formulation can also be applied in settings with time-varying constraints by replacing $h(x_k)$ with a time-dependent barrier function $h(x_k, k)$. This extension to time-varying constraints does not mitigate the dependence on the horizon or the limited expressiveness of the parametrization.

## 4-2 Neural Network CBF

**Set of Learnable Parameters:** $\{W_i, b_i, \ell_\theta, \ell_{f,\theta}\}$. The learnable parameters $W_i$ and $b_i$ are the weight matrices and biases of the neural network $\text{NN}_\theta(\cdot)$ respectively, which outputs the decay rates $\gamma_i$ used in the CBF constraint. The functions $\ell_\theta$ and $\ell_{f,\theta}$ are the stage cost and terminal cost in the MPC objective (see Section 2-1-1).

### Single CBF Constraint Formulation

Even though LOPTD-CBF adapts decay rates online and as such enhances feasibility, performance is limited by strong dependence on the MPC horizon and by a restricted parametrization that cannot capture richer state-dependent safety conditions. An alternative that is less dependent on the horizon and more expressive removes the optimal-decay decision variable and uses a neural network to output the decay rate $\gamma$ in the discrete CBF condition (2-30):

$$h(x_{k+1}) - h(x_k) \geq -\gamma\, h(x_k) \quad \rightarrow \quad h(x_{k+1}) - h(x_k) \geq -\mathrm{NN}_\theta(\cdot) h(x_k). \tag{4-3}$$

The neural network takes as input the current state $x_k$ and the CBF value $h(x_k)$ at each time step $k$, making the decay rate a state dependent function. The network can then be represented as follows:

$$\begin{aligned} z^{(0)} &= \begin{bmatrix} x_k \\ h(x_k) \end{bmatrix}, \\ z^{(i)} &= \mathrm{ReLU}(W_i\, z^{(i-1)} + b_i), \quad i = 1, \ldots, L, \\ \gamma &= \mathrm{Sigmoid}(W_{L+1}\, z^{(L)} + b_{L+1}). \end{aligned}$$

A schematic of this neural network is shown in Figure 4-1 below.

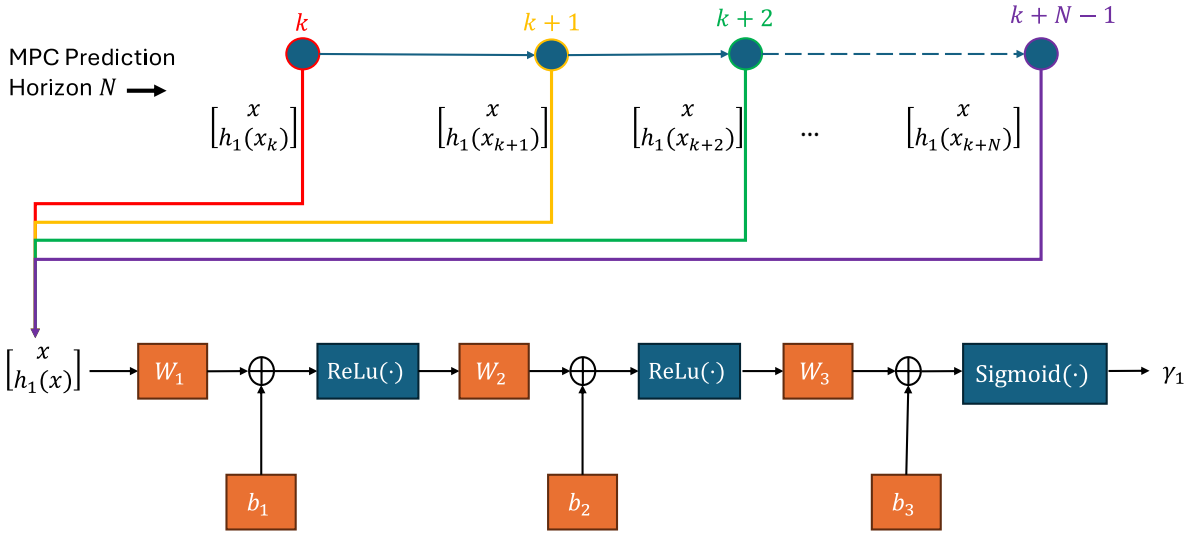



**Figure 4-1:** NN-CBF neural network architecture.

Using this network in the CBF constraint yields the following parameterized MPC value function $V_\theta(s)$:

$$
\begin{aligned}
V_\theta(s) = \min_{X,U,\Sigma} \quad & \ell_{f,\theta}(x_N) + \sum_{k=0}^{N-1} \ell_\theta(x_k,\, u_k) + w_{\mathrm{MPC}} \sum_{k=0}^{N-1} \sigma_k && \text{(4-4a)}\\
\text{s.t.} \quad & x_{k+1} = f(x_k,\, u_k), \qquad k = 0, \ldots, N-1, && \text{(4-4b)}\\
& x_k \in \mathcal{S}, \qquad k = 0, \ldots, N, && \text{(4-4c)}\\
& u_k \in \mathcal{A}, \qquad k = 0, \ldots, N-1, && \text{(4-4d)}\\
& x_0 = s, && \text{(4-4e)}\\
& h(x_{k+1}) - \left(1 - \mathrm{NN}_\theta(x_k, h(x_k))\, h(x_k)\right) \geq -\sigma_k, \qquad k = 0, \ldots, N-1, && \text{(4-4f)}\\
& \sigma_k \;\geq\; 0, \qquad k = 0, \ldots, N-1. && \text{(4-4g)}
\end{aligned}
$$

Compared with the LOPTD-CBF, this method is less dependent on the MPC horizon. This reduced dependence is especially useful for short horizons, since RL can still learn effective policies while keeping MPC computational overhead low. Moreover, the NN further enables richer nonlinear mappings, allowing finer trade-offs between safety and performance. However, reliance on a NN typically increases the number of trainable parameters relative to the optimal-decay method. It also introduces topological and hyperparameter choices that require tuning.

#### Multiple CBF Constraints Formulation

Extending this framework to multiple CBFs requires only a minor modification. The input vector is augmented to include the barrier values of all active CBFs. Let $\mathcal{O}$ denote the total number of CBFs, and let $i = 1, \ldots, \mathcal{O}$ index them. Then

$$
z^{(0)} = \begin{bmatrix} x_k \\ h_1(x_k) \\ \vdots \\ h_{\mathcal{O}}(x_k) \end{bmatrix}. \tag{4-5}
$$

The final layer of the network is adjusted to produce one decay rate $\gamma_i$ per CBF as follows:

$$
\begin{bmatrix} \gamma_1 \\ \vdots \\ \gamma_{\mathcal{O}} \end{bmatrix} = \mathrm{Sigmoid}(W_{L+1}, z^{(L)} + b_{L+1}). \tag{4-6}
$$

Figure 4-2 depicts this expanded network architecture.

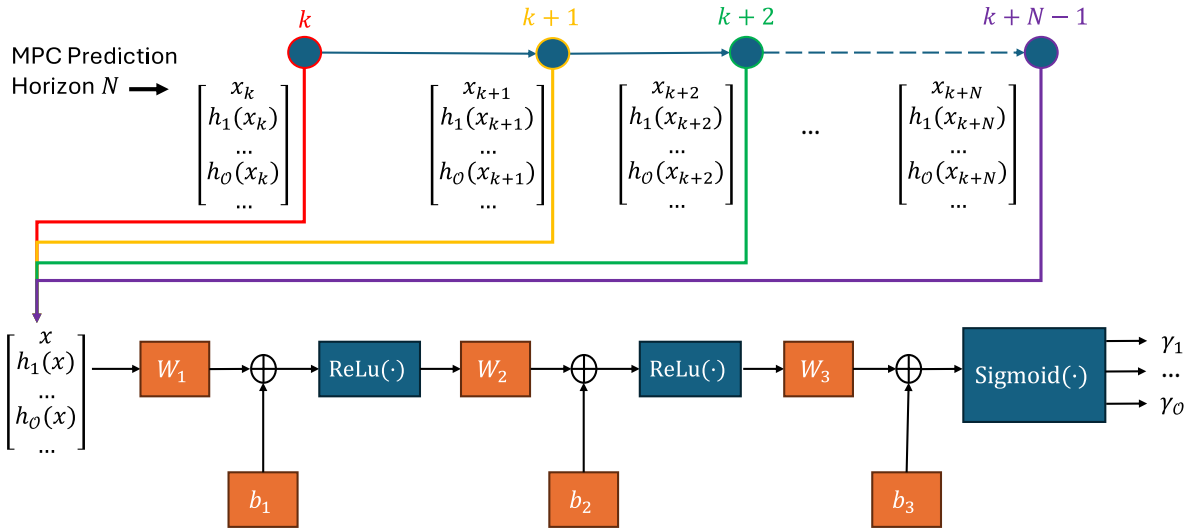



**Figure 4-2:** NN-CBF neural network architecture extended to handle multiple CBF constraints.

This formulation can also handle time-varying CBFs $h(x_k, k)$, such as those arising from moving obstacles or other dynamic constraints. In this case, the network input can be augmented with additional contextual information that may aid decision-making. This information, encoded at time step $k$ as the context $c_i(k)$ for each CBF $i = 1, \ldots, \mathcal{O}$, can represent any relevant variable associated with the constraint. The augmented input vector is then defined as

$$z_k^{(0)} = \begin{bmatrix} x_k & h_1(x_k, k) \cdots h_{\mathcal{O}}(x_k, k) & c_1(k) \cdots c_{\mathcal{O}}(k) \end{bmatrix}^\top,$$

In an obstacle avoidance setting, for instance, the context $c_i(k)$ may correspond to the position of obstacle $i$, written in 1D as its scalar location or, in 2D, as $c_i(k) = [x_i(k),\, y_i(k)]^\top$ and stacked accordingly in the NN input. This augmentation provides additional directional or contextual information that the scalar barrier functions $h_i(x_k, k)$ alone cannot capture, allowing the network to better distinguish, for example, whether an obstacle lies to the left or right of the system. Including such context can improve performance, but it comes at the cost of higher input dimensionality, more trainable parameters, and increased computational load. The resulting MPC formulation for the NN-CBF is then described as follows.

$$V_\theta(s) = \min_{X, U, \Sigma} \quad \ell_{f,\theta}(x_N) + \sum_{k=0}^{N-1} \ell_\theta(x_k,\, u_k) + w_{\text{MPC}} \sum_{k=0}^{N-1} \sum_{i=1}^{\mathcal{O}} \sigma_{k,i} \tag{4-7a}$$

$$\text{s.t.} \quad x_{k+1} = f(x_k,\, u_k), \qquad k = 0, \ldots, N-1, \tag{4-7b}$$

$$x_k \in \mathcal{S}, \qquad k = 0, \ldots, N, \tag{4-7c}$$

$$u_k \in \mathcal{A}, \qquad k = 0, \ldots, N-1, \tag{4-7d}$$

$$x_0 = s, \tag{4-7e}$$

$$h(x_{k+1}) - \left(1 - \text{NN}_\theta(x_k, h_i(x_k), c_i(k))\, h(x_k)\right) \geq -\sigma_{k,i}, \quad \begin{matrix} k = 0, \ldots, N-1, \\ i = 1, \ldots, \mathcal{O}, \end{matrix} \tag{4-7f}$$

$$\sigma_{k,i} \;\geq\; 0, \quad \begin{matrix} k = 0, \ldots, N-1, \\ i = 1, \ldots, \mathcal{O}. \end{matrix} \tag{4-7g}$$

## 4-3 Recurrent Neural Network CBF

**Set of Learnable Parameters:** $\{W_i, b_i, W_{q^{(i)}}, \ell_\theta, \ell_{f,\theta}\}$ The learnable parameters are the weight matrices $W_i$, the biases $b_i$ and the recurrent weights $W_{q^{(i)}}$ of the RNN. Together they define $\text{RNN}_\theta(\cdot)$, which outputs the decay rates $\gamma_i$ used in the CBF constraints. The functions $\ell_\theta$ and $\ell_{f,\theta}$ are the stage cost and terminal cost in the MPC objective (see Section 2-1-1).

Building on the previous feedforward formulation, an Elman recurrent neural network (RNN) [17] is used to generate decay rates over prediction the horizon. The previous NN-based CBF condition is then reformulated by replacing the NN with an RNN:

$$h_i(x_{k+1}) - h_i(x_k) \geq -\text{NN}_\theta(\cdot)h_i(x_k) \quad \rightarrow \quad h_i(x_{k+1}) - h_i(x_k) \geq -\text{RNN}_\theta(\cdot)h_i(x_k). \tag{4-8}$$

The RNN is defined by the following equations:

$$\begin{aligned}
z_k^{(0)} &= \begin{bmatrix} x_k & h_1(x_k,k)\cdots h_{\mathcal{O}}(x_k,k) & c_1(k)\cdots c_{\mathcal{O}}(k) \end{bmatrix}^\top, \\
q_k^{(i)} &= \text{ReLU}(W_i\, z_k^{(0)} + b_i + W_{q^{(i)}} q_{k-1}^{(i)}), \quad i = 1, \\
q_k^{(i)} &= \text{ReLU}(W_i\, q_k^{(i-1)} + b_i + W_{q^{(i)}} q_{k-1}^{(i)}), \quad i = 2, \ldots, L, \\
\begin{bmatrix} \gamma_1 \\ \vdots \\ \gamma_{\mathcal{O}} \end{bmatrix} &= \text{Sigmoid}(W_{L+1}\, q_k^{(L)} + b_{L+1}),
\end{aligned}$$

where $q_k^{(i)}$ denotes the hidden state of layer $i$ at time step $k$, and the full recurrent neural network architecture is shown in Figure 4-3.

The key idea is that RNNs store past information in their hidden state. This allows it to "remember" recent pieces of context information, something a feedforward network does not explicitly capture. As a result, the RNN is able to handle time-varying CBFs more effectively, since its structure is designed to exploit temporal relation. For example, it can potentially store in memory recent obstacle trajectories and velocities through the hidden state. By storing temporal information in the hidden state, RNNs can also train more sample-efficiently than feedforward NNs in tasks involving dynamic systems [11]. A sufficiently large feedforward NN can, in theory, approximate any mapping, including those that depend on past inputs. However, achieving this typically requires more training samples and is more difficult than when such temporal structure is inherently imposed, as in the RNN.

In addition to acting as a memory element, the RNN is rolled out across the MPC horizon by updating its hidden state at each predicted step with the corresponding predicted input, as illustrated in Figure 4-3. At every horizon step, the predicted state is fed into the RNN, which outputs the $\gamma$ value used in the discrete CBF condition. The hidden state is then passed recursively through the architecture, so that each future step has access not only to past trajectory information but also to the sequence of predicted inputs along the horizon. In this way, a point at horizon step $k+5$ receives information given by the input at $k+4$, $k+3$, $k+2$, and so on, through the evolving hidden state. This ensures that the CBF conditions at different steps of the horizon are temporally connected through the RNN. Once the MPC

optimization is solved and the first control action $u_0^\star$ is applied, the RNN hidden state is reset to its value before the optimization, $q_{k-1}$. It is then updated using the new actual state $s_{k+1}$, so that the hidden state becomes consistent with where the system has truly moved. This guarantees that the hidden state follows the real trajectory of the system rather than the predicted one.

The RNN can be similarly embedded into a parametrized MPC $V_\theta(s)$, as done for the NN-CBF (4-7).

$$V_\theta(s) = \min_{X,U,\Sigma} \quad \ell_{f,\theta}(x_N) + \sum_{k=0}^{N-1} \ell_\theta(x_k,\, u_k) + w_{\text{MPC}} \sum_{k=0}^{N-1} \sum_{i=1}^{\mathcal{O}} \sigma_{k,i} \tag{4-9a}$$

$$\text{s.t.} \quad x_{k+1} = f(x_k,\, u_k), \qquad k = 0, \dots, N-1, \tag{4-9b}$$

$$x_k \in \mathcal{S}, \qquad k = 0, \dots, N, \tag{4-9c}$$

$$u_k \in \mathcal{A}, \qquad k = 0, \dots, N-1, \tag{4-9d}$$

$$x_0 = s, \tag{4-9e}$$

$$h(x_{k+1}) - (1 - \text{RNN}_\theta(x_k, h_i(x_k), c_i(k))\, h(x_k)) \geq -\sigma_{k,i}, \quad \begin{matrix} k = 0, \dots, N-1, \\ i = 1, \dots, \mathcal{O}, \end{matrix} \tag{4-9f}$$

$$\sigma_{k,i} \;\geq\; 0, \quad \begin{matrix} k = 0, \dots, N-1, \\ i = 1, \dots, \mathcal{O}. \end{matrix} \tag{4-9g}$$

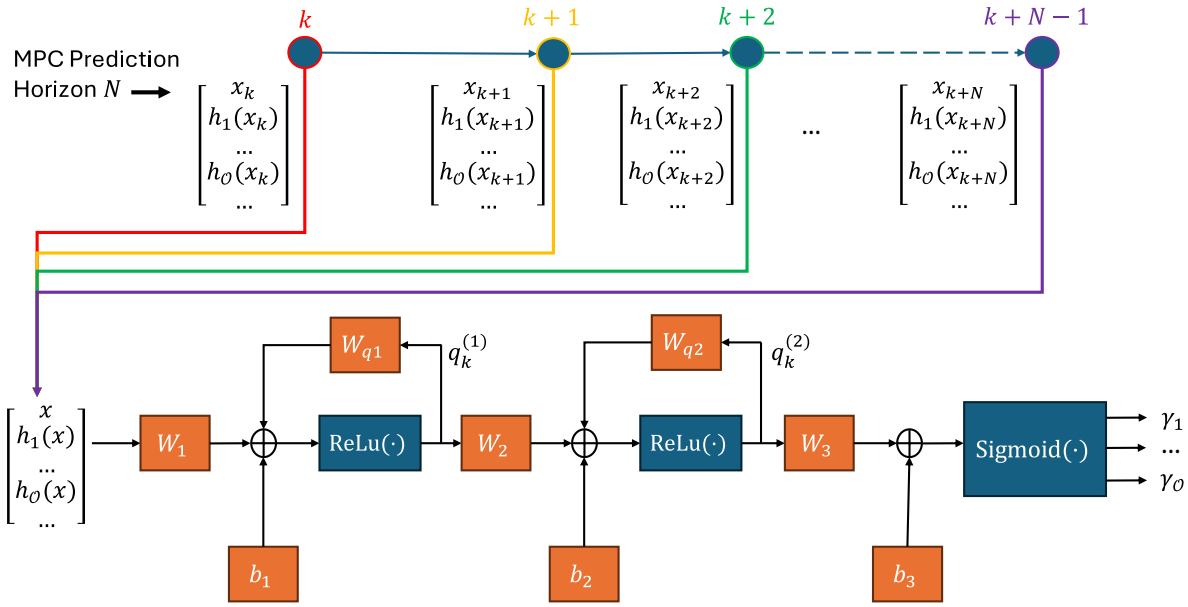



**Figure 4-3:** RNN-CBF recurrent neural network architecture.

## 4-4 Training Architecture

The overall training architecture for the methods in this thesis follows the structure in Algorithm 1, where $N_{\text{episodes}}$ denotes the total number of training episodes. This RL training

process is structured around three MPCs. The first MPC is $V_{rand,\theta}(s,\xi)$, a state value function that provides the policy during training. To induce exploration, this MPC adds a perturbation term $\xi^\top u_0$ to its objective, where $\xi$ is drawn from a normal distribution. Injecting noise through the MPC keeps the constraints satisfied and, hence, is preferred to exploration methods in which the control action is directly perturbed as $a = u_0 + \xi_t$, which can lead to constraints violations. Furthermore, the noise level $\xi$ decays during training so the policy shifts from exploration to exploitation. Lastly, the second and third MPCs compute $V_\theta(s)$ and $Q_\theta(s,a)$ used to form the TD error in (2-15).

An integral part of the training architecture is the RL stage cost $L(s,a)$, which encodes the return of the task at hand as per eq. (2-3). In this training architecture, the stage cost is augmented to include the effect of the slack variables used by the MPC controller as follows.

$$L_{\text{aug}}(s_k, a_k, \Sigma_k^\star) = L(s_k, a_k) + w_{\text{RL}} \sum_{k=0}^{N-1} \sum_{i=1}^{\mathcal{O}} \sigma_{k,i}^\star \tag{4-10}$$

Here, the RL stage cost $L(s,a)$ and can be chosen to be similar to the MPC stage cost $\ell_\theta(s,a)$ so that RL stage cost remains aligned with the MPC objective. The second term aggregates the optimal slack variables $\Sigma^\star$ from the optimal solution of the MPC $V_{\text{rand},\theta}(s_k,\xi)$. The weight $w_{RL}$ sets how strongly violations are penalized during learning. These weights can in principle be different from the weights introduced in the MPC formulations e.g. (4-4).

Using the previously defined Q-learning update (2-16), a per-step update $g_i = -\tau_i \nabla_\theta Q_\theta(s_i,a_i)$ is computed and stored in a buffer. Once the buffer is full, the stored updates are averaged to form

$$\mathbf{g}_{\text{avg}} = \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} g_i, \tag{4-11}$$

where $|\mathcal{B}|$ denotes the buffer cardinality. The parameter update follows an adjusted version of (2-16) that replaces plain gradient descent with an Adam optimizer

$$\theta \leftarrow \theta - \mathbf{g}_{\text{Adam}}, \quad \mathbf{g}_{\text{Adam}} = \text{Adam}(\mathbf{g}_{avg}, \theta), \tag{4-12}$$

where the Adam algorithm is described in Appendix B.

However, these parameter updates must also respect parameter bounds when required. For example, if the MPC stage cost $\ell_\theta(x_k,u_k)$ must remain positive definite, the relevant parameters must be bounded to preserve this property. To allow both lower and upper bounds, the update step is cast into the following QP, as done in [5]:

$$\Delta\boldsymbol{\theta}^\star = \arg\min_{\Delta\boldsymbol{\theta}} \quad \frac{1}{2}\Delta\boldsymbol{\theta}^\top \Delta\boldsymbol{\theta} + \alpha \mathbf{g}_{\text{Adam}}^\top \Delta\boldsymbol{\theta} \tag{4-13a}$$

$$\text{s.t.} \quad \boldsymbol{\theta}_{\text{lb}} \le \boldsymbol{\theta} + \Delta\boldsymbol{\theta} \le \boldsymbol{\theta}_{\text{ub}}, \tag{4-13b}$$

$$\Delta\boldsymbol{\theta}_{\text{lb}} \le \Delta\boldsymbol{\theta} \le \Delta\boldsymbol{\theta}_{\text{ub}}, \tag{4-13c}$$

where $\Delta\boldsymbol{\theta}^\star$ is the projection of $-\alpha\mathbf{g}_{\text{Adam}}^\top$ onto the bounded parameter space. The bounds in the QP are given by $\boldsymbol{\theta}_{\text{lb}}$ and $\boldsymbol{\theta}_{\text{ub}}$, while $\Delta\boldsymbol{\theta}_{\text{lb}}$ and $\Delta\boldsymbol{\theta}_{\text{ub}}$ limits the rate of change. The parameter update is then given by $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}^\star$. In Algorithm 1 the Adam update together with the QP projection is represented by one step called $\text{AdamQP}(\boldsymbol{\theta}, \mathbf{g}_{avg})$. For further details of the code see the GitHub Repository.

**Algorithm 1** RL Training Loop

**Require:** Learning parameters $\theta = [P_\theta, \cdots]$.

**Require:** RL hyperparameter (see Appendix A) including learning rate $\eta$, discount factor $\gamma_{RL}$, noise schedule, buffer size, patience threshold, and update frequency.

**for** episode = 1 to $N_{episode}$ **do**
  Reset the environment
  Reset RNN hidden states (if RNN is used)
  **for** $t = 0$ to $T - 1$ **do**
    **(a) Exploration**
    Sample noise $\xi_t$ and solve the noisy policy MPC $V_{\text{rand},\theta}(s_t, \xi_t)$ using one of:
      - LOPTD–CBF: (4-2), with $+\xi_t^\top u_0$ added to objective,
      - SNN–CBF: (4-7), with $+\xi_t^\top u_0$ added to objective,
      - SRNN–CBF: (4-9), with $+\xi_t^\top u_0$ added to objective.

    and obtain the exploratory action $a_t = u_0^\star$ and optimal slacks $\Sigma_t^\star$
    **(b) System rollout**
    Apply action $a$ and update the system state $s_{t+1} \leftarrow f(s_t, a_t)$
    Observe stage cost $L_{\text{aug}}(s_t, a_t, \Sigma_t^\star)$ (4-10)
    **(c) Q-value**
    Solve $Q_\theta(s_t, a_t)$ MPC using one of:
      - LOPTD–CBF: (4-2), with the constraint $u_0 = a_t$ added,
      - SNN–CBF: (4-7), with the constraint $u_0 = a_t$ added,
      - SRNN–CBF: (4-9), with the constraint $u_0 = a_t$ added,

    and obtain optimal primal-dual variables $y^\star$
    **(d) V-value**
    Solve $V_\theta(s_{t+1})$ MPC using one of:
      - LOPTD–CBF: (4-2),
      - SNN–CBF: (4-7),
      - SRNN–CBF: (4-9).

    **(e) Temporal-difference error**
    Compute $\tau_t \leftarrow L_{\text{aug}}(s_t, a_t, \Sigma_t^\star) + \gamma_{\text{RL}} V_\theta(s_{t+1}) - Q_\theta(s_t, a_t)$
    **(f) Gradient computation**
    Compute sensitivity $\nabla_\theta Q_\theta(s_t, a_t)$ using $\nabla_\theta \mathcal{L}_\theta(s_t, a_t, y^\star)$ as outlined in section 2-2-3
    Form the update $\mathbf{g}_t \leftarrow -\tau_t \nabla_\theta Q_\theta(s_t, a_t)$
    Store $\mathbf{g}_t$ in buffer $\mathcal{B}$
  **end for**
  **if** buffer $\mathcal{B}$ is full **then**
    Take mean over buffer $\mathbf{g}_{avg} \leftarrow \frac{1}{|\mathcal{B}|} \sum \mathbf{g}_t$ and clear the buffer $\mathcal{B}$
    Update $\theta \leftarrow \text{AdamQP}(\theta, \mathbf{g}_{avg})$
  **end if**
  Decay exploration noise ($\xi_t$)
**end for**

# Chapter 5

# Simulation Results

The results in this chapter are split into two parts. The first part of this chapter reports the viability of the LOPTD-CBF and NN-CBF methods on a simple example with one static obstacle. The second part of this chapter explores a more complex environment with multiple moving obstacles, where the RNN-CBF and NN-CBF are tested against each other to compare the RNN with the NN in this scenario. Both experiments use the following discrete linear 2D double-integrator system, the same one as given in [38]:

$$s_{k+1} = As_k + Ba_k, \tag{5-1}$$

$$A = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 \\ 0 & \frac{1}{2}\Delta t^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix}, \tag{5-2}$$

where the sampling time $\Delta t$ is set to 0.2s.

The system is controlled by an MPC-CBF where the avoidance of these obstacles shares the following MPC terminal and stage cost

$$x_N^T P\, x_N + \sum_{k=0}^{N-1} (x_k^T Q\, x_k \; + \; u_k^T R\, u_k), \tag{5-3}$$

where $Q = 10 \cdot \mathcal{I}_4, R = \mathcal{I}_2, P = 100 \cdot \mathcal{I}_4$. Additionally, the system is subject to the following state and input constraints: $\mathcal{S} = \{s_k \in \mathbb{R}^n : s_{\min} \leq s_k \leq s_{\max}\}, \mathcal{A} = \{s_k \in \mathbb{R}^m : s_{\min} \leq s_k \leq$ with the lower and upper bounds being

$$s_{\max},\, s_{\min} = \pm 5\, \mathbf{I}_{4\times 1}, \quad s_{\max},\, s_{\min} = \pm \mathbf{I}_{2\times 1}. \tag{5-4}$$

The RL stage cost is chosen as follows, with the same reasoning as outlined earlier in 4-4.

$$L_{\text{aug}}(s, a, \Sigma^\star) = s^\top Q s + a^\top R a + w_{RL} \sum_{k=0}^{N-1} \sum_{i=1}^{\mathcal{O}} \sigma_{k,i}^\star. \tag{5-5}$$

Further details of the RL training algorithm parameters for the different experiments and the bounds of the learnable parameters are provided in Appendix A.

## 5-1 Static Obstacle

As mentioned before, the experiment for the first part involved avoiding one circular static obstacle, as shown in Figure 5-1. The obstacle is described by $h(s) = (s_0 - 2)^2 + (s_1 - 2.25)^2 - 1.5^2$ , where everything outside of the obstacle is the safe set $\mathcal{C} = \{s \in \mathcal{S} \mid h(s) \geq 0\}$. Because the task was relatively simple, the MPC controller was implemented with a prediction horizon of one. This deliberately myopic design was intended to produce a sub-optimal policy that could later be improved upon.

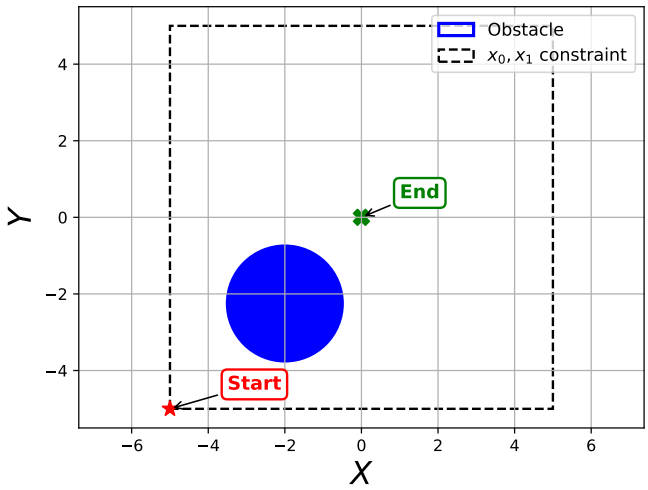



**Figure 5-1:** Static Obstacle Experiment Setup

In the results of the static obstacle experiment presented below, the part of the MPC formulation that is parameterized is the terminal cost, excluding the parameters related to the CBF condition. The terminal cost matrix is constrained to remain positive definite by enforcing lower bounds greater than zero during the update step, as detailed in Section 4-4. The exact parameter bounds are listed in Appendix A.

To provide a point of comparison for the learned policies in this subsection, a reference optimal policy was calculated. This was achieved by using a fixed non-parametrized MPC which replaced the CBF constraint with $h(x_k) \geq 0$ and was simulated with a prediction horizon of $N = 100$. This resulted in the policy shown in Figure 5-2, which achieves a total cumulative RL stage cost of **5475**, when evaluated with the same RL stage cost function used throughout (5-5) this chapter.

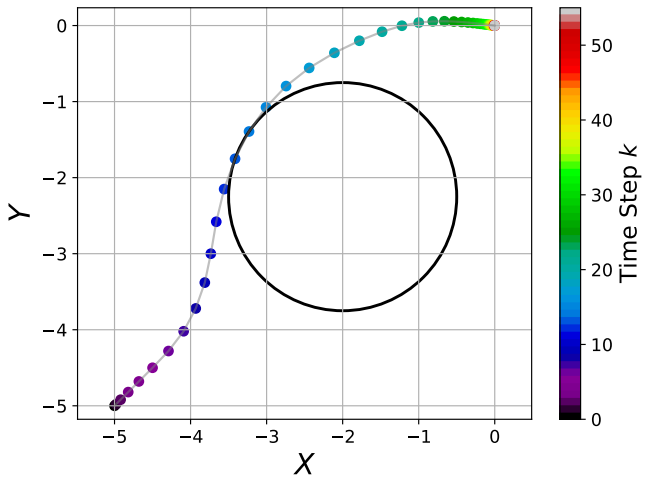



**Figure 5-2:** Reference Optimal Trajectory for the Experiment

### 5-1-1 Learnable Optimal Decay CBF

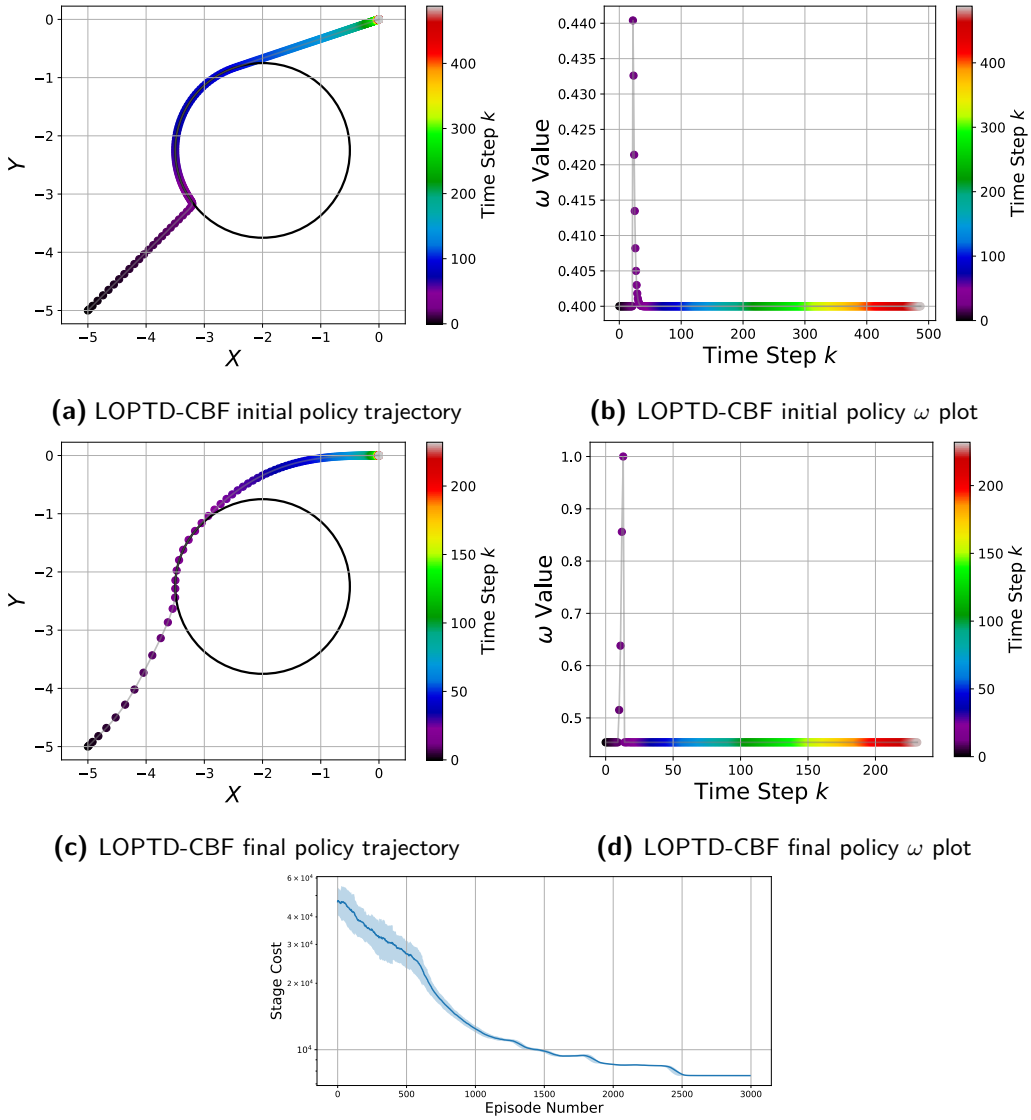
Experiments were first conducted with the LOPTD-CBF. Before training, the initial policy was highly suboptimal, as shown in Figure 5-3a, resulting in a cumulative stage cost of **21712**. The one-step-horizon MPC-CBF makes the policy very myopic, preventing it from accounting for the obstacle ahead. Consequently, the system trajectory moves diagonally toward the goal until it reaches the obstacle boundary, at which point it turns and follows the boundary to maintain safety and satisfy the CBF condition.

The point at which the trajectory reaches the obstacle boundary is also evident in Figure 5-3b, which shows the evolution of the decision variable $\omega$. The decision variable $\omega$, introduced in Section 2-3-6, serves as an adaptive decay rate in the discrete eCBF condition, where the MPC optimizes $\omega$ at each step instead of keeping the decay rate fixed. The decay rate is detailed in Section 2-3-5 and its influence on MPC trajectories is illustrated in Figure 2-5 in that section. For most of the trajectory, $\omega$ remains close to 0.4. A distinct spike appears once the trajectory encounters the boundary. This spike enables the trajectory to turn left and follow the obstacle boundary, as a larger $\omega$ relaxes the CBF constraint just enough to keep the condition feasible while permitting a deviation from the straight path. Because the system velocity is low, the spike is relatively small, meaning the constraint is only slightly relaxed during the turn. By contrast, at higher system velocities, one would expect larger changes in $h(s_{k+1}) - h(s_k)$, which would necessitate greater spikes in $\omega$ values.

After training, Figure 5-3e shows that the cumulative stage cost per episode decreased, with the final value reduced to **7156**. This improvement is also evident in Figure 5-3c, where the trajectory no longer moves directly toward the goal, but instead anticipates the obstacle and steers left. The trajectory reaches the target more quickly due to a higher velocity, which was reflected in the reduced number of iterations required to arrive at the goal. Nevertheless, the trajectory still incurs a higher cumulative stage cost than the optimal trajectory in Figure 5-2 and requires more iterations to reach the origin.

The effect of the improved policy is also apparent in the $\omega$ dynamics. In Figure 5-3d, the $\omega$ value again exhibits a spike, but this time it reaches a substantially higher $\omega$ value. The

larger spike indicates the need for greater relaxation of the CBF constraint near the obstacle, since higher velocities lead to larger variations in the CBF $h(x_k)$. In other words, because the MPC now drives the system at higher velocity, maintaining feasibility of the CBF constraint requires a larger decay rate $\omega$.




**(a)** LOPTD-CBF initial policy trajectory

**(b)** LOPTD-CBF initial policy $\omega$ plot

**(c)** LOPTD-CBF final policy trajectory

**(d)** LOPTD-CBF final policy $\omega$ plot

**(e)** Cumulative stage cost across training episodes, with running mean (blue line) and running standard deviation (shaded region) computed using a centered 100-episode window.

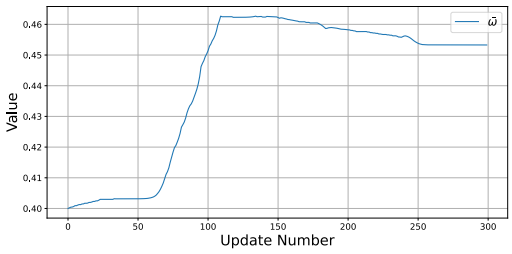
**Figure 5-3:** Policy before and after training

The improved policy can be directly linked to changes in the learned parameters. Figure 5-4c shows the evolution of the four diagonal entries of the learned terminal cost matrix during training. This terminal cost matrix $P$ is the same as the one introduced in the MPC objective

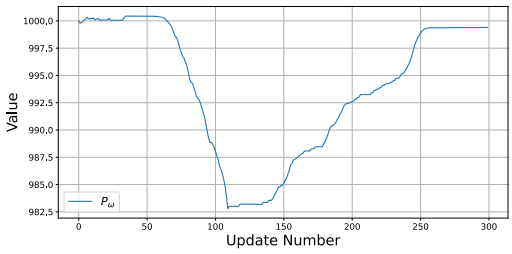
at the start of this chapter (5-3). It defines the terminal cost $\ell_f$, which shapes how terminal states are weighted and is used in the methods presented in Chapter 4. In the figure, the blue and orange curves correspond to $P_{1,1}$ and $P_{2,2}$, which weight the terminal $x$ and $y$ positions. The green and red curves correspond to $P_{3,3}$ and $P_{4,4}$, which weight the terminal $x$ and $y$ velocities.

When the position weights $P_{1,1}$ and $P_{2,2}$ dominate the velocity weights $P_{3,3}$ and $P_{4,4}$, the resulting trajectories become faster. This is expected, since applying a stronger terminal penalty on positions than on velocities means positions not at the origin are penalized more heavily than nonzero velocities. As a result, the MPC favors reaching the target quickly, accepting higher velocities because they incur smaller penalties, while strongly penalizing failure to reach the origin. The two velocity weights are also not equal. $P_{3,3}$ has a higher value than $P_{4,4}$, which penalizes x-velocity more than y-velocity. In other words, velocities along the y-axis are punished less than those along the x-axis. The optimizer therefore prefers to move along the y-axis to incur lower cost, effectively steering left. This behavior matches the learned policy.

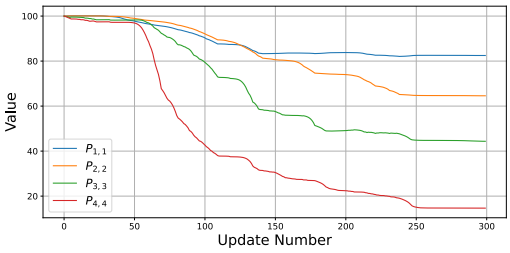
In addition, the evolution of $\bar{\omega}$ and $P_\omega$ is shown in Figure 5-4a and Figure 5-4b, respectively. The role of these variables in the LOPTD-CBF framework is explained and summarized in Section 4-1. The value of $P_\omega$ shown in the figure remains largely unchanged, while $\bar{\omega}$ increases slightly during training. A larger $\bar{\omega}$ value corresponds to a larger decay rate and therefore a less conservative policy. The initial value $P_\omega = 1000$ was selected after experimenting with nearby values, since smaller initial values caused the RL agent to reduce $P_\omega$ toward 0. If $P_\omega$ becomes too small, $\bar{\omega}$ is no longer restricted by the solver and will likely tend to 1 to make the CBF condition less conservative. Reducing conservatism too early can backfire, since the system may approach the obstacle too quickly and later fail to find a feasible action that guarantees safety.




**(a)** Evolution of $\bar{\omega}$ during training.




**(b)** Evolution of $P_\omega$ during training.




**(c)** Evolution of terminal cost matrix $P$ during training.

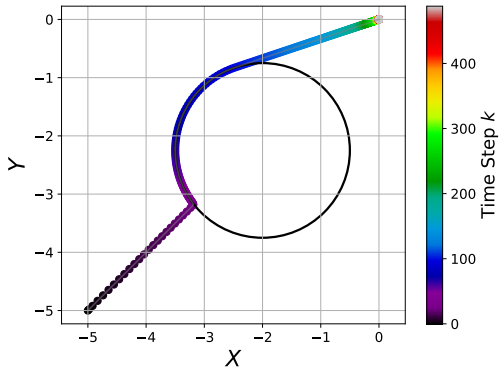
**Figure 5-4:** Evolution of learned parameters during training. Update number corresponds to one parameter update every 10 episodes, over a total of 3000 episodes.

### 5-1-2 Neural Network CBF

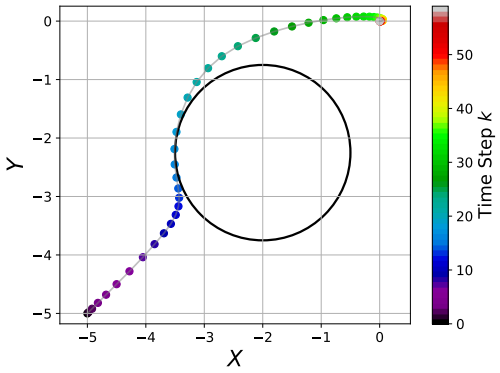
After evaluating the LOPTD-CBF, experiments were conducted with the NN-CBF. The initial policy of this MPC was again suboptimal, yielding a cumulative stage cost of **21892**, as shown in Figure 5-5a. The trajectory closely resembles the case discussed in the previous subsection, where the one-step-horizon MPC-CBF is too myopic to anticipate the obstacle.

Following training, the cumulative stage cost decreased to **6627**, as shown in Figure 5-5d. This value is lower than the cumulative stage cost achieved with the LOPTD-CBF but remains higher than that of the optimal trajectory. The final improved policy is shown in Figure 5-5b, where the trajectory turns using the CBF safety condition rather than relying solely on the terminal cost. To further analyze this behavior, the evolution of the decay rate $\gamma$ value (the NN output) is shown in Figure 5-5c.

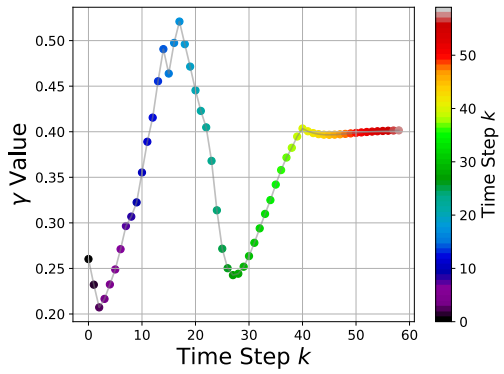
Initially, $\gamma$ decreases (purple dots at the beginning), making the policy more conservative and preventing the system from accelerating too quickly. It then increases, allowing faster progress toward the target. Near the obstacle, $\gamma$ decreases again (light blue dot), introducing conservatism that enables a safe turn around the circle. After passing the obstacle, $\gamma$ increases once more to permit faster motion toward the target. Finally, as the system approaches the goal, $\gamma$ decreases again, ensuring that the trajectory slows smoothly and avoids overshooting. Afterward, it stabilizes, allowing the system to settle at the target.



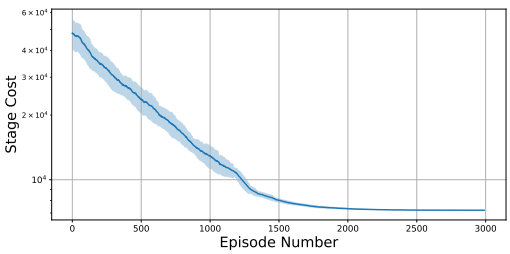

**(a)** NN-CBF initial policy trajectory.




**(b)** NN-CBF final policy trajectory.




**(c)** NN-CBF final policy $\gamma$ plot.




**(d)** Cumulative stage cost across training episodes, with running mean (blue line) and running standard deviation (shaded region) computed using a centered 100-episode window.

**Figure 5-5:** Policy before and after training.

Looking at the evolution of the terminal cost parameters in Figure 5-6, a decrease in the velocity weights relative to the position weights allows the system to move faster, similar to the LOPTD-CBF case. In this case, however, the gap between position and velocity weights is even larger, which further increases the system's speed and enables it to reach the target in fewer iterations. At the same time, the two velocity weights remain close to each other, unlike in the LOPTD-CBF case where the $x$-velocity was penalized more than the $y$-velocity. This equal penalty on both $x$ and $y$ velocities drives the system to move forward rather than steer left. Consequently, the turn made by the trajectory is more likely governed by the CBF condition than by the terminal cost.

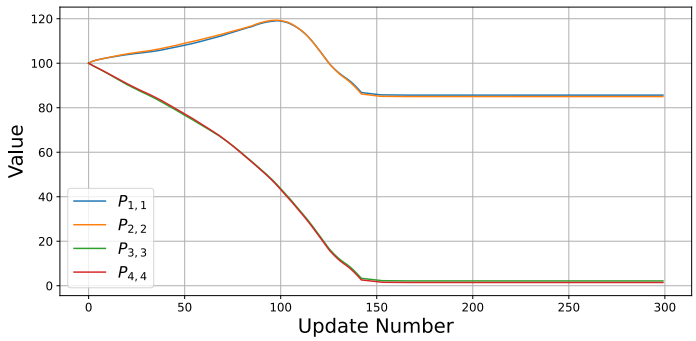



**Figure 5-6:** Evolution of terminal cost matrix $P$ during training. Update number corresponds to one parameter update every 10 episodes, over a total of 3000 episodes.

### 5-1-3 Comparison

Both the LOPTD-CBF and the NN-CBF improve over the initial MPC-CBF policy, each with their own advantages and limitations. Table 5-1 highlights the main trade-offs between the two approaches.

The LOPTD-CBF requires fewer parameters than the NN-CBF, which makes training generally faster because there are fewer degrees of freedom for the RL agent to optimize. Furthermore, the parameters over which the LOPTD-CBF learns have more interpretability than the learned NN parameters. NN interpretability still holds at the output level, since the decay rate values given by the NN output can be interpreted. Moreover, because the LOPTD-CBF includes the decay rate $\omega$ as a decision variable in the MPC, it can adjust the trajectory and safety online. This makes the configuration safer during training, with less chance of using slacks in ways that incur safety violations. However, its performance is strongly dependent on the MPC horizon and with a short horizon it has limited freedom to balance safety against performance.

In contrast, the NN-CBF employs a larger number of trainable parameters, which allows the network to capture more complex safety functions and generate state-dependent decay rates. Owing to its richer parameterization, the MPC with the NN-CBF is less dependent on the horizon length for tuning the decay rate compared to the LOPTD-CBF. This combination of reduced dependence on the horizon and the ability to represent more complex safety functions is reflected in the results above, where the final cumulative stage cost is smaller than that achieved with the LOPTD-CBF, indicating better overall performance. However, the NN is

less likely to guarantee safety during training. During episode rollouts, the network produces fixed decay rates determined by its pre-trained parameters, and these cannot be adjusted online, unlike the LOPTD-CBF, which can adapt its decay rate dynamically within each rollout. For example, when approaching an obstacle at high velocity, the network cannot dynamically choose to increase its decay rate but must use the value already given by the NN. This issue is especially pronounced when the network is poorly initialized.

| **Property** | **LOPTD-CBF** | **NN-CBF** |
|---|---|---|
| Fewer parameters | X | |
| Interpretability | X | |
| Not dependent on MPC horizon | | X |
| More degrees of freedom to balance safety | | X |
| Less chance of safety violations during training | X | |
| Final cumulative stage cost | **7156** | **6627** |

**Table 5-1:** Comparison of properties between LOPTD-CBF and NN-CBF

Additionally, in this simple example the terminal cost matrix $P$ plays a pivotal role in shaping the system trajectory. To test its importance, the experiments were repeated without learning $P$, with results shown in Appendix C. The results show that in the absence of a learnable terminal cost matrix, the RL agent does not achieve an improved policy for either method. Since, the CBF condition by itself cannot accelerate the trajectory toward the target and therefore the cost does not decrease.

Lastly, even though both methods improved over the initial policy, they were still unable to recover the optimal policy shown in Figure 5-2. This indicates that there is still room for improvement, which could potentially be achieved through a richer parametrization, for example by also parameterizing the $Q$ and $R$ matrices in (5-3).

## 5-2 Dynamic Obstacles

Extending the framework to account for time-varying obstacles required the design of a new experiment. The setup of this experiment is shown in Figure 5-7. In this figure, two dynamic obstacles move horizontally to hinder the system from reaching the target. Their motion follows the Step Bounce model, which is defined as follows.

At $k = 0$, each obstacle is initialized with a position $c_0^{\text{obs}} \in [c_{\min}, c_{\max}]$ and a direction $d_0 \in \{-1, +1\}$, where $+1$ corresponds to motion to the right and $-1$ to motion to the left. The initial direction of the obstacle is indicated by the arrow in Figure 5-7. The obstacle moves with constant speed $v > 0$, which remains fixed for the entire experiment. At each sampling step $\Delta t$, the candidate position is updated according to

$$\tilde{c}_{k+1} = c_k^{\text{obs}} + d_k v \Delta t. \tag{5-6}$$

If $\tilde{c}_{k+1} \in [c_{\min}, c_{\max}]$, then the update is accepted with $c_{k+1}^{\text{obs}} = \tilde{c}_{k+1}$ and $d_{k+1} = d_k$. Otherwise, the position is reflected back into the admissible interval and the direction reverses:

$$(c_{k+1}^{\text{obs}}, d_{k+1}) = \begin{cases} (2c_{\max} - \tilde{c}_{k+1},\ -d_k), & \tilde{c}_{k+1} > c_{\max}, \\ (2c_{\min} - \tilde{c}_{k+1},\ -d_k), & \tilde{c}_{k+1} < c_{\min}. \end{cases} \tag{5-7}$$

In addition to these dynamic obstacles, a static obstacle is placed above them, forcing the system to find a path between the two moving obstacles. This obstacle is represented by the green circle in Figure 5-7. The complete obstacle configuration is provided in Table A-9 in the Appendix. Based on this setup, the CBFs for the obstacles are defined as

$$\begin{aligned} h(s,k) &= (s_0 - c_k^{\text{obs}_1})^2 + (s_1 + 1.5)^2 - 0.7^2, \quad c_0^{\text{obs}_1} = -2,\ d_0 = 1,\ v = 2.3, \\ h(s,k) &= (s_0 - c_k^{\text{obs}_2})^2 + (s_1 + 3.3)^2 - 0.7^2, \quad c_0^{\text{obs}_2} = -3,\ d_0 = -1,\ v = 2.0, \\ h(s,k) &= (s_0 + 2)^2 + (s_1 - 0)^2 - 1^2. \end{aligned} \tag{5-8}$$

The first CBF corresponds to the first dynamic obstacle shown in blue in Figure 5-7 and is described by the Step Bounce model. This obstacle initially moves to the right and has a higher velocity than the other dynamic obstacle. The second dynamic obstacle, also described by the Step Bounce model, is shown in orange and corresponds to the second dynamical CBF. It initially moves in the opposite direction, to the left. The third CBF represents the static obstacle, indicated in green in the figure.

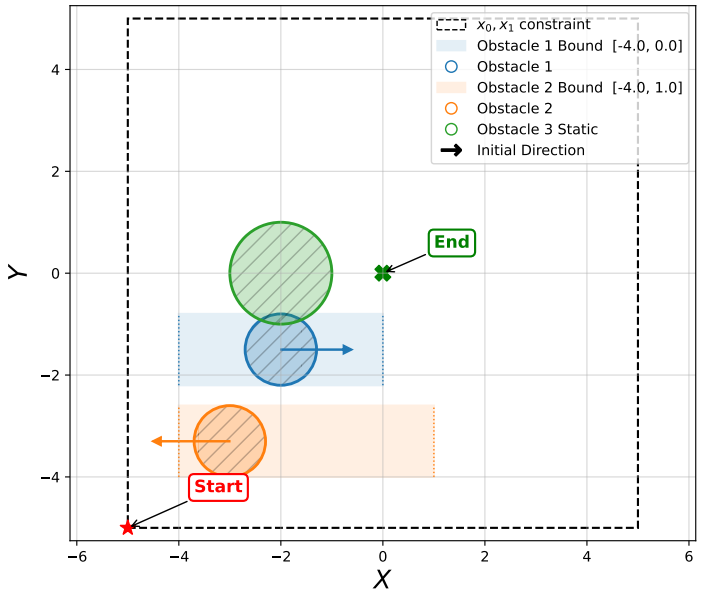



**Figure 5-7:** Dynbamic Obstacles Experiment Setup

The experiments in this section were conducted both with and without the terminal cost matrix $P$ parameterization. Since the resulting performance and trajectories were highly similar, the results with the parameterized terminal cost are omitted. The results shown in this section correspond to the experiments without terminal cost $P$ parameterization. The similarity between the two configurations is explained later in this thesis.

Once again, as in Section 5-1, an optimal policy was computed to provide a point of comparison for the learned policies in this subsection. This policy was obtained using an MPC that replaced the CBF constraint with $h(s_k) \geq 0$ and simulated it with a prediction horizon of $N = 100$. The resulting trajectory, shown in Figure 5-8, includes snapshots of the system and the moving obstacle. When evaluated with the same RL stage cost function used throughout this section, the policy achieves a total cumulative cost of **4933**.

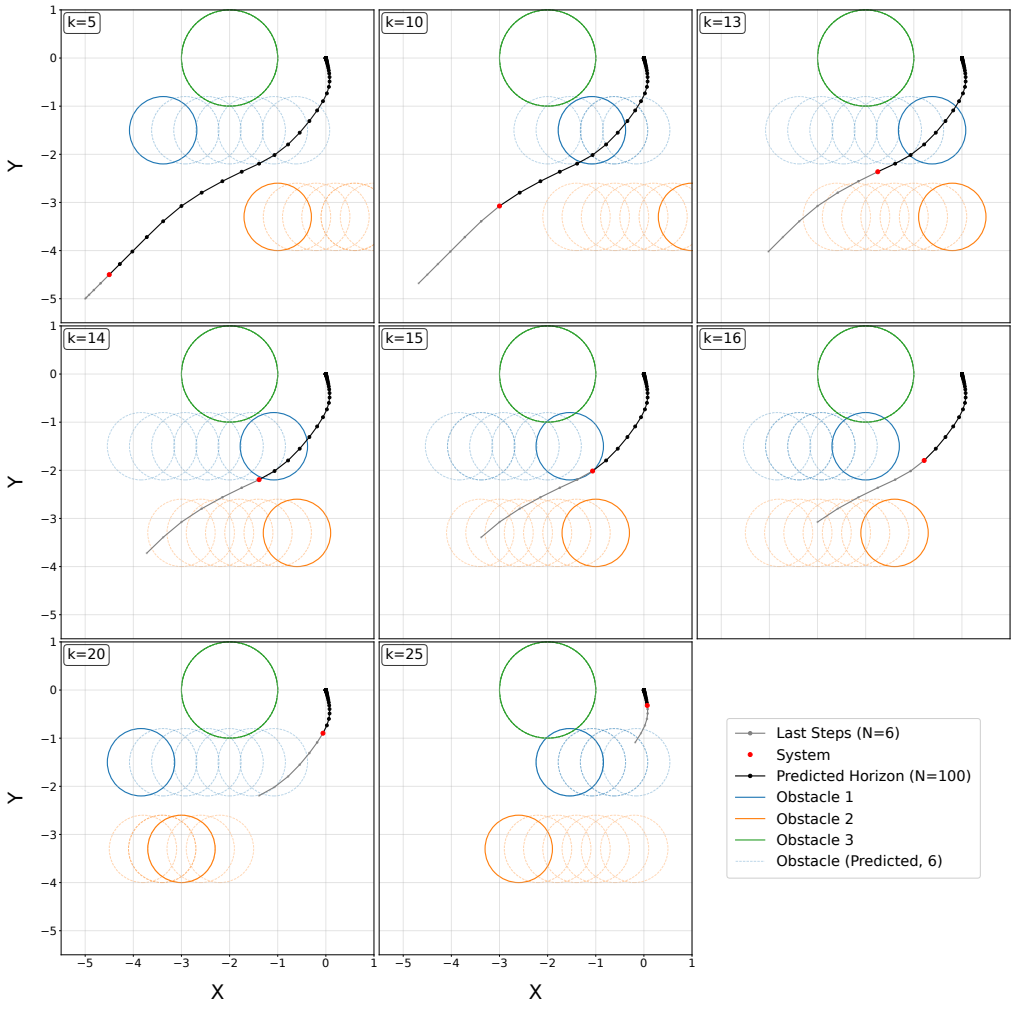



**Figure 5-8:** Snapshots of the reference optimal trajectory obtained with MPC ($N = 100$). The red dot marks the current system position, the black trail shows the predicted horizon and the grey line shows the last six steps of the system trajectory. The solid circles indicate the current positions of the static (green) and dynamic (blue and orange) obstacles, while the dotted circles indicate their predicted positions six time steps ahead.

Lastly, because the initialization of the networks influences the final learned policy, the weights of the feed-forward layers in both networks were initialized within a smaller range. This ensured that the initial output (the decay rate) started close to a flat value of 0.5. Such initialization provides a fairer basis for comparing the two methods and helps generate a less-performing initial policy, giving the RL agent the opportunity to learn and improve upon it.

### 5-2-1 Neural Network CBF

Starting with the initial policy before learning, shown in Figure 5-9, the system trajectory is unsafe, as the system enters the obstacle at $k = 16$ and $k = 17$. Moreover, the predicted steps of the MPC also fail to avoid the predicted obstacles. This behavior results from the suboptimal $\gamma$ outputs of the RNN, shown in Figure 5-11a.

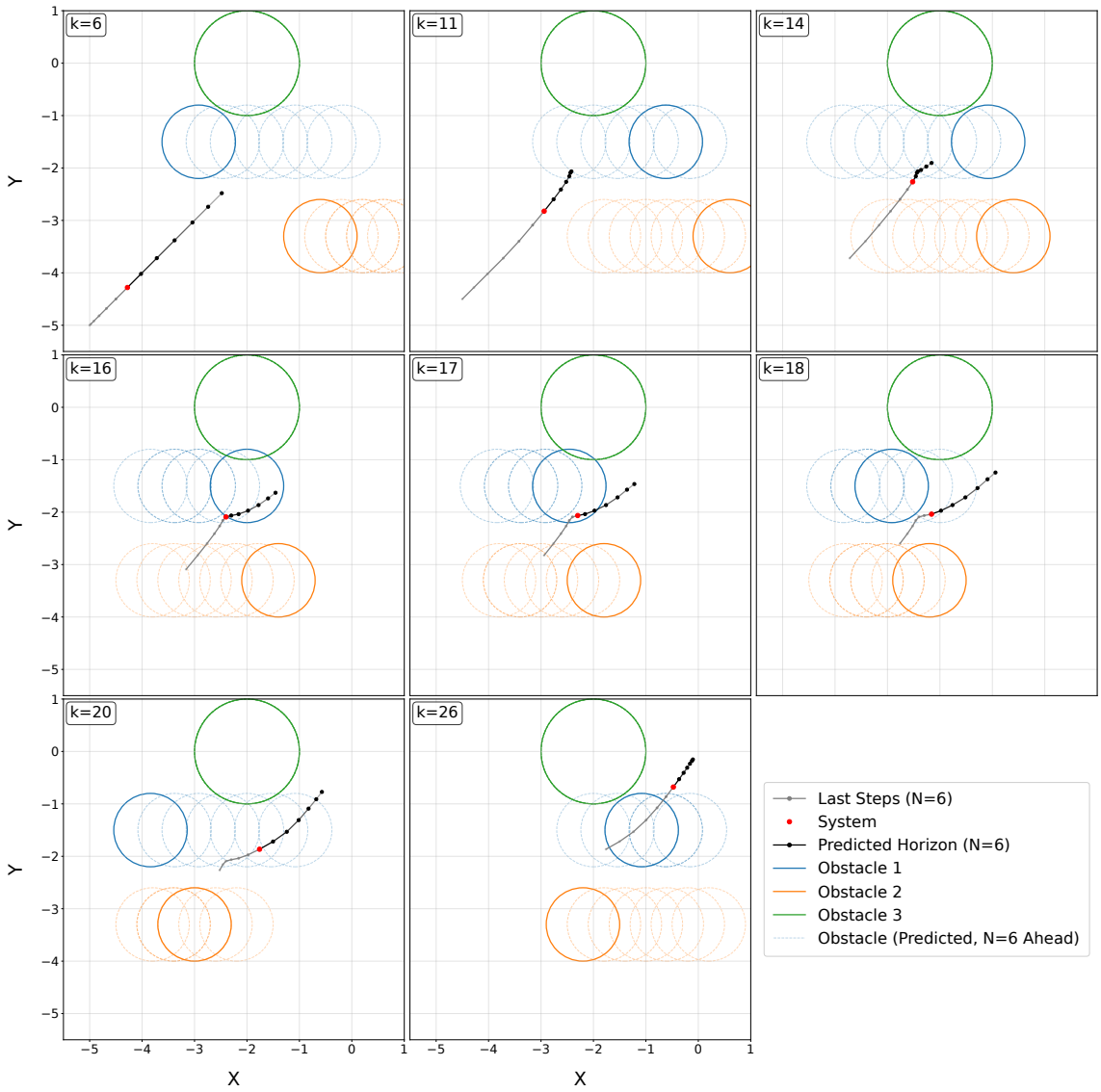



**Figure 5-9:** Snapshots of the initial policy before learning. The red dot marks the current system position, the orange trail shows the predicted horizon and the blue line shows the last six steps of the system trajectory. The solid circles indicate the current positions of the static (green) and dynamic (blue and orange) obstacles, while the dotted circles indicate their predicted positions six time steps ahead.

After training, the improved trajectory is shown in Figure 5-10. The trajectory now slows down earlier to avoid the first dynamic obstacle (blue). At $k = 16$, $k = 17$, and $k = 18$, the

system remains outside this obstacle and maneuvers around it, indicating that it has learned a safe trajectory. This behavior is also reflected in the predicted horizon, where the predicted trajectory avoids the future positions of the first dynamic obstacle (blue), demonstrating that the system plans safely ahead.

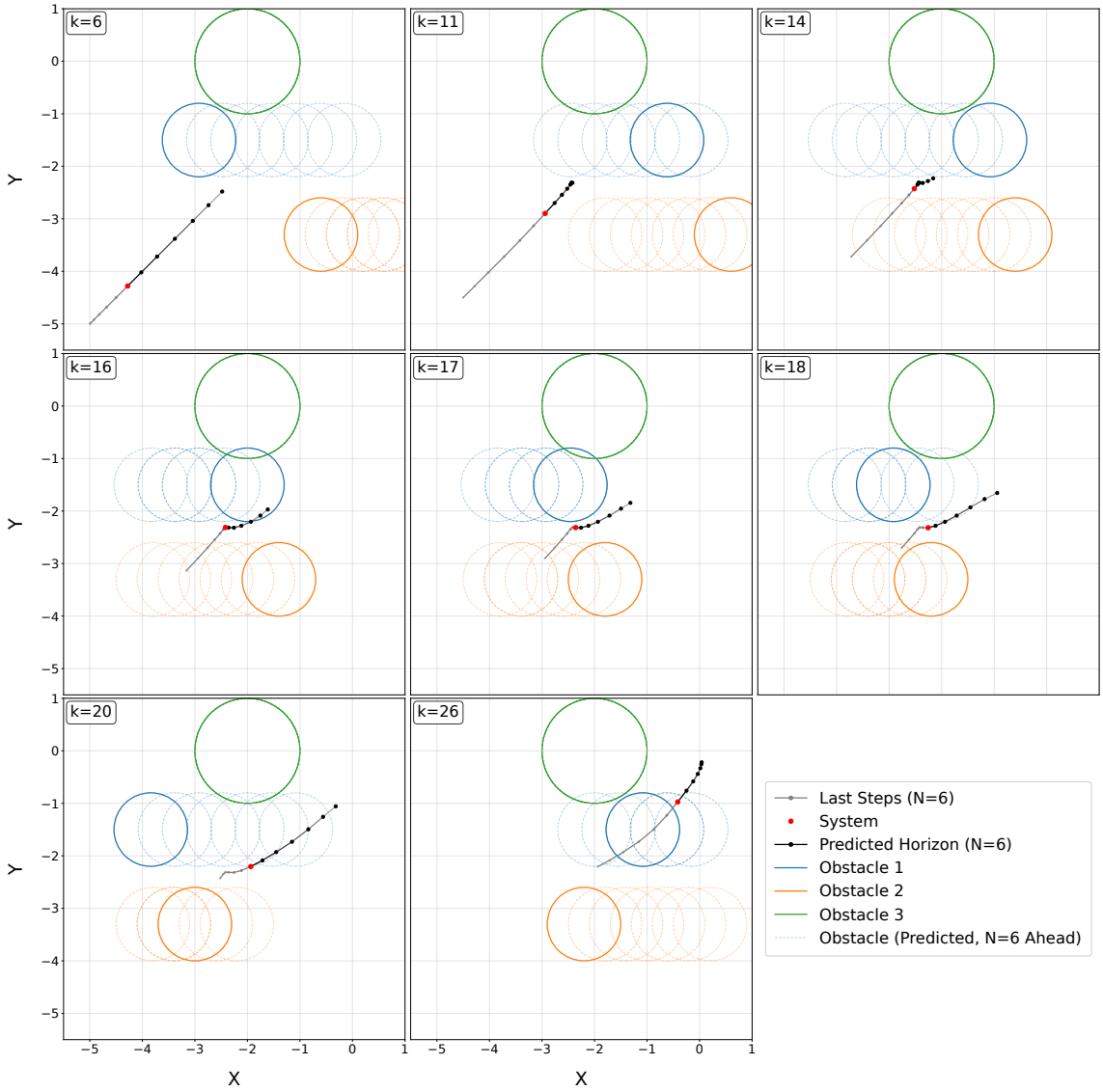
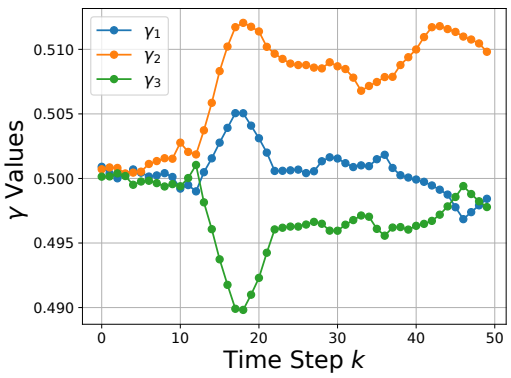



**Figure 5-10:** Snapshots of the improved policy after training. The red dot marks the current system position, the black trail shows the predicted horizon and the grey line shows the last six steps of the system trajectory. The solid circles indicate the current positions of the static (green) and dynamic (blue and orange) obstacles, while the dotted circles indicate their predicted positions six time steps ahead.

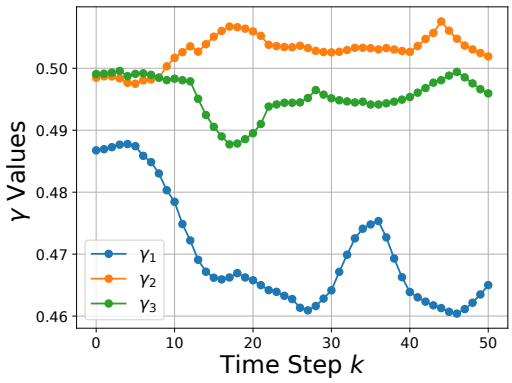
This improvement is also reflected in the NN outputs shown in Figure 5-11b. The $\gamma_1$ values for the first dynamic obstacle (blue) change the most, which is expected since this was the obstacle whose constraint was previously violated. The change appears as a decrease in the decay rate after training, with a lower decay rate corresponding to a more conservative

approach (see Section 2-3-5 and Figure 2-5 for a detailed explanation of the decay rate). This added conservativeness allows the system to slow down earlier when approaching the obstacle. The most pronounced drop in $\gamma_1$ occurs between $k = 8$ and $k = 15$, directly linking to the behavioral change in the trajectory. At $k = 11$, for example, the predicted horizon in Figure 5-9 still enters the predicted obstacle, whereas in Figure 5-10 the system slows down much earlier, enabling it to bypass the obstacle safely.

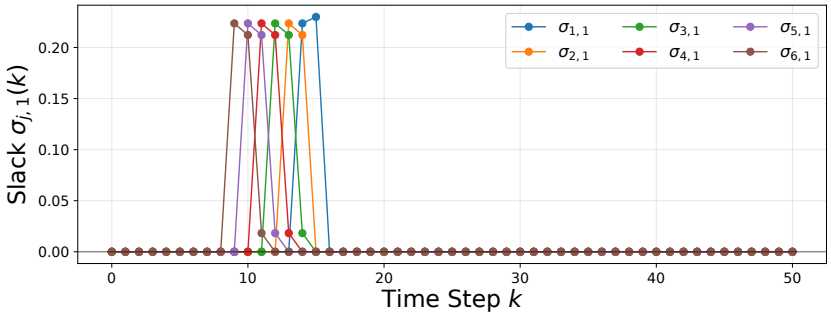
However, achieving this behavior still requires the use of slack variables, as shown in Figure 5-11c. As explained in Chapter 4, slacks are introduced during training to temporarily allow constraint violations so that the CBF condition remains feasible. At the same time, penalties on slack usage encourage the MPC to minimize violations and gradually learn a safe, control-invariant solution. Ideally, the network would learn a safe policy without relying on slacks, but this was not achieved. Importantly, in the solution illustrated in Figure 5-10, the slack variables are not used to permit violations but rather to allow the system to decelerate quickly enough under bounded control inputs. This becomes especially relevant when obstacles move at higher velocities or when the control input bounds are tight. While such a policy without slack usage is theoretically possible, it was not demonstrated in this numerical experiment. Nevertheless, the results remain valid since the policies, even if not strictly control-invariant, still yield safe trajectories. The stage cost of the trajectory shown in Figure 5-10 is **6067** when slack penalties are not included and **15194** when they are accounted for, as also shown in comparison with the RNN results in Table 5-2.




**(a)** $\gamma$ values of the NN before training (initial policy).




**(b)** $\gamma$ values of the NN after training (final policy).




**(c)** MPC slack variables used across horizon points for Obstacle 1 (blue)

**Figure 5-11:** NN outputs and MPC slacks over policy iterations. Panels (a) and (b) show NN $\gamma$ values before and after training. Panel (c) shows slack variables produced by the MPC.

### 5-2-2 Recurrent Neural Network CBF

The initial policy for the RNN-CBF follows the same pattern as in the NN-CBF case. The $\gamma$ outputs start flat at around 0.5, and the trajectory is unsafe at $k = 16$ and $k = 17$, where the system enters the obstacle, as illustrated in Figure 5-12.

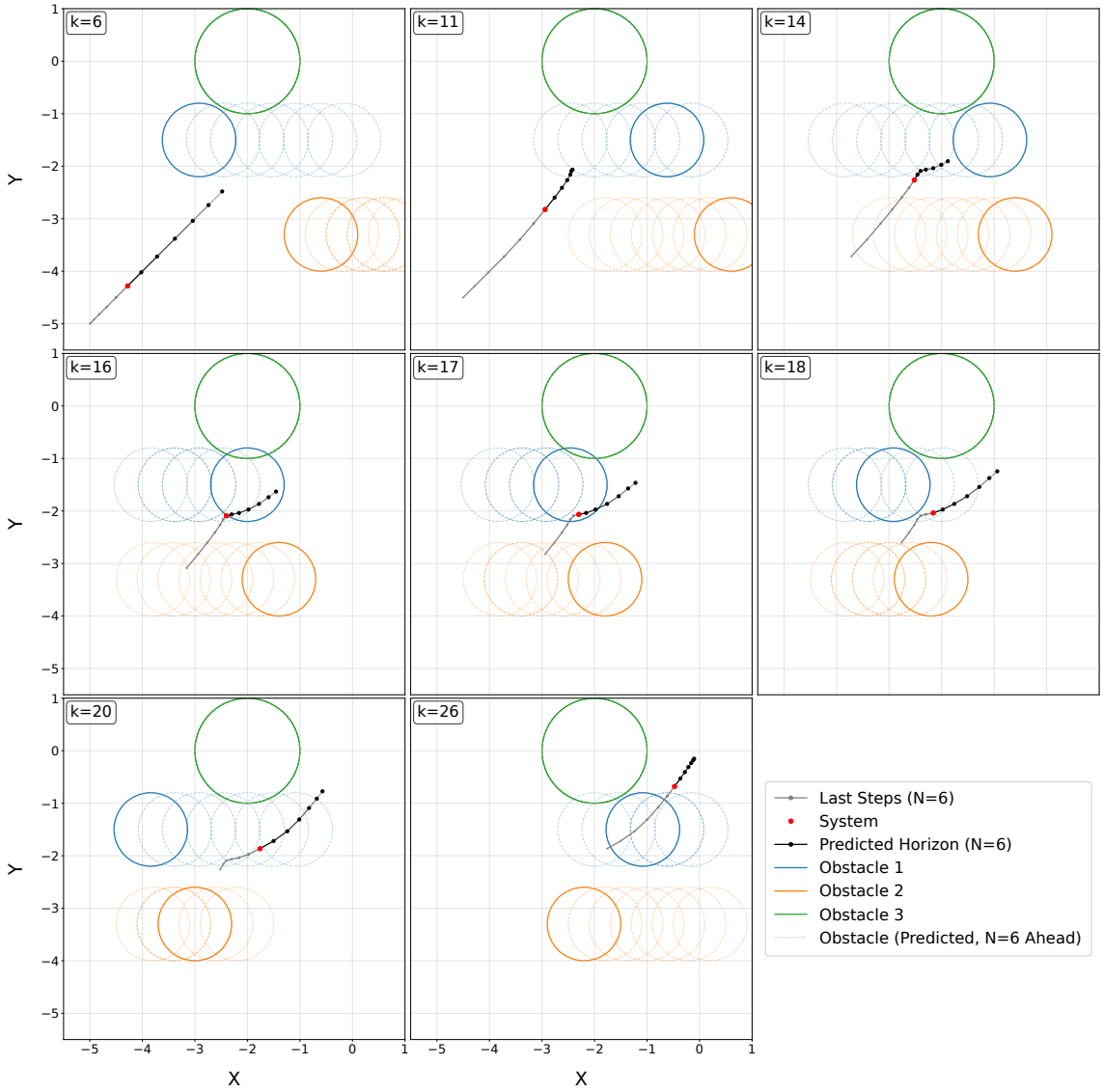



**Figure 5-12:** Snapshots of the initial RNN-CBF policy before training. The red dot marks the current system position, the orange trail shows the predicted horizon and the blue line shows the last six steps of the system trajectory. The solid circles indicate the current positions of the static (green) and dynamic (blue and orange) obstacles, while the dotted circles indicate their predicted positions six time steps ahead.

After training, the improved trajectory is shown in Figure 5-13. A similar safe policy is achieved as in the NN-CBF case, though it is less conservative, as the trajectory comes closer to the obstacles at $k = 17$.

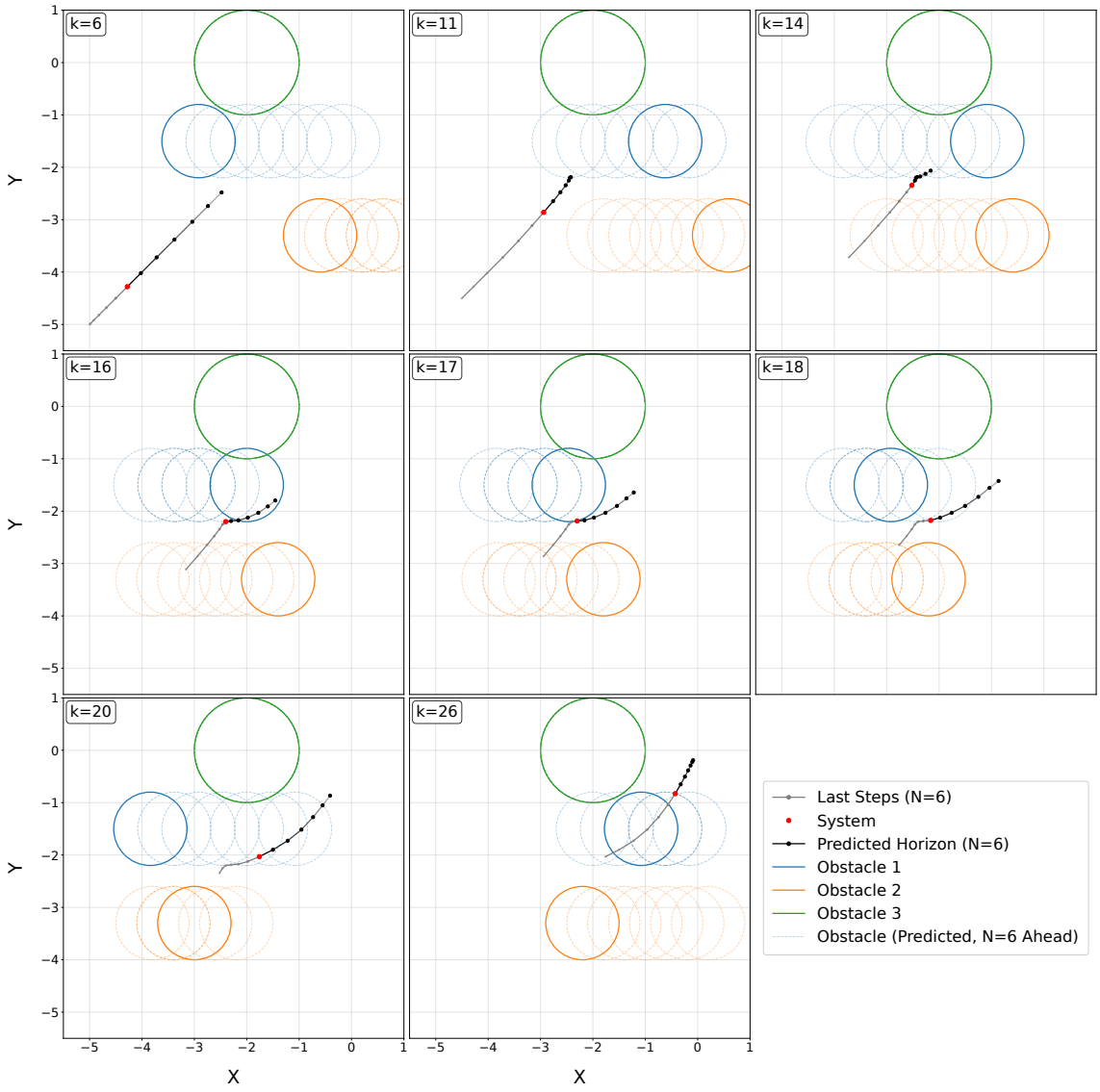
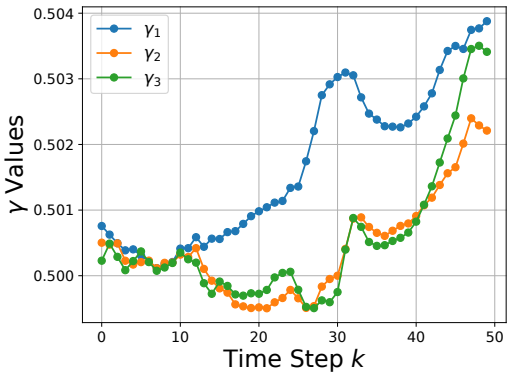


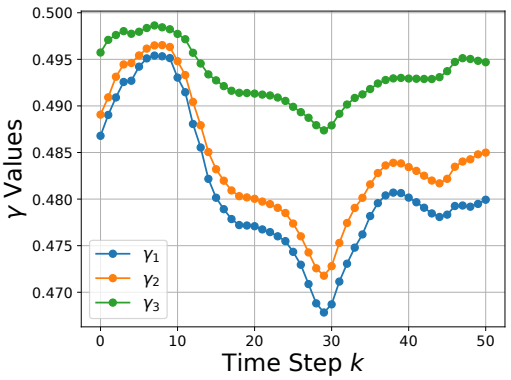

**Figure 5-13:** Snapshots of the improved RNN-CBF policy after training. The red dot marks the current system position, the black trail shows the predicted horizon and the grey line shows the last six steps of the system trajectory. The solid circles indicate the current positions of the static (green) and dynamic (blue and orange) obstacles, while the dotted circles indicate their predicted positions six time steps ahead.

The output of the RNN follows the same trend as in the NN case. The $\gamma_1$ values for the first dynamic obstacle (blue), shown in Figure 5-13, decrease between $k = 8$ and $k = 15$. This decrease is smaller than in the NN case, meaning the decay rate remains slightly higher and results in less conservative behavior, as seen in the trajectory at $k = 16$. However, slack variables are again required to achieve adequate deceleration, as shown in Figure 5-14c. The same reasoning applies as in the NN-CBF case. That is, with a richer RNN parametrization, this behavior could potentially be learned without the use of slacks. The stage cost of the trajectory in Figure 5-13 is **5923** when slack penalties are omitted and **14026** when they are included, as shown in comparison with the NN results in Table 5-2. Together, these results

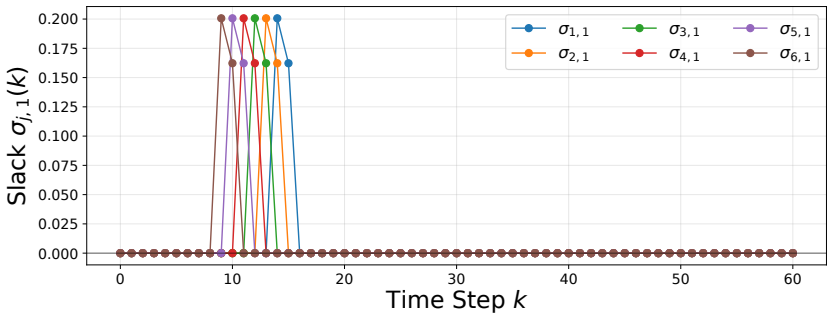
indicate that both the NN and the RNN are able to learn comparable policies.




**(a)** $\gamma$ values before training (initial policy).




**(b)** $\gamma$ values after training (final policy).




**(c)** MPC slack variables used across horizon points for Obstacle 1 (blue).

**Figure 5-14:** RNN outputs and MPC slacks over policy iterations. Panels (a) and (b) show $\gamma$ values before and after training, while panel (c) shows slack variables produced by the MPC for Obstacle 1.

With extended training, the RNN can learn a policy that avoids both obstacles without relying on slacks, instead taking a longer trajectory. While this provides a slack-free solution, it comes at the expense of a higher RL and MPC stage cost when slacks are not considered. By reducing the learning rate earlier, the slack-using policy can be preserved, whereas further training enables the RNN to converge to this alternative slack-free solution. This result is illustrated in Appendix D.

### 5-2-3 Comparison

Comparing the two policies of the NN-CBF and RNN-CBF, one aspect stands out: the time required to learn the policy. As shown in Figure 5-15, the RNN achieves the same policy with about 10 fewer evaluations. This corresponds to roughly 170 fewer training episodes in total, demonstrating that the RNN requires less training time thanks to its recurrent structure. Further experiments with different setups and configurations are needed to confirm this finding, as it has been tested only in this numerical experiment and as such the claim cannot yet be made conclusively. Nevertheless, the current result indicates that the RNN is able to learn a policy faster than the NN.

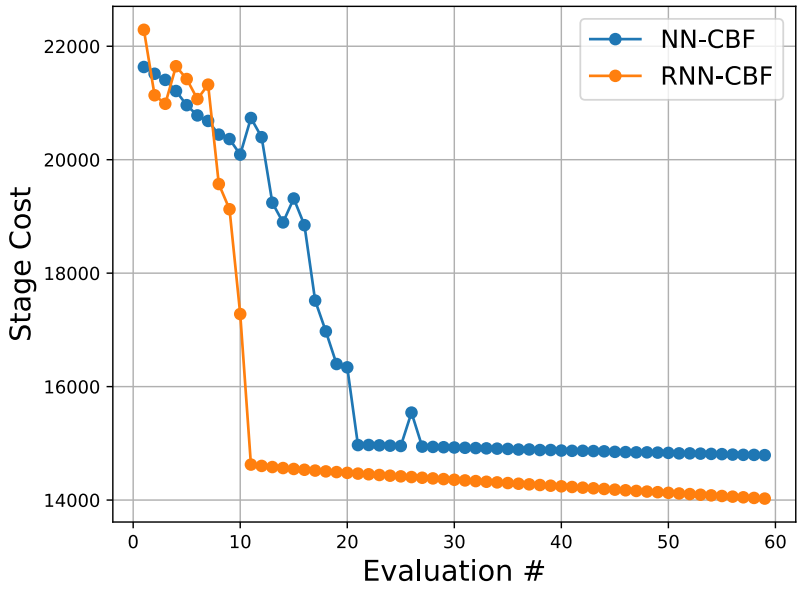



**Figure 5-15:** Validation stage cost comparison between NN-CBF and RNN-CBF across evaluation steps. Evaluations are performed every 50 updates during training, where validation means testing the policy at that point without the exploratory noise $\xi^\top u_0$.

Additionally, the trajectory of the RNN is more effective, as it follows a policy that comes closer to the obstacles and therefore achieves a lower stage cost, as seen in Table 5-2. It also relies slightly less on slack variables, although the overall slack usage remains comparable, as shown in Figure 5-11c for the NN-CBF and Figure 5-14c for the RNN-CBF.

| **Final Cumulative Stage Cost** | **NN-CBF** | **RNN-CBF** |
|---|---|---|
| With Slacks | **15194** | **14026** |
| Without Slacks | **6067** | **5923** |

**Table 5-2:** Comparison of results between NN-CBF and RNN-CBF

It is important to note that both the NN-CBF and RNN-CBF are able to find a solution without using the MPC terminal cost matrix $P$, as defined in (5-3). Training with a terminal cost yields a very similar solution, so the corresponding results are omitted, as they closely resemble those in Figure 5-9 and Figure 5-13 with a comparable RL stage cost. The similar performance is likely due to the increased prediction horizon, which provides more opportunities for parameter tuning. With a longer horizon, more CBF constraints are introduced at each step, giving both the NN and RNN additional flexibility to adjust the trajectory by tuning a larger number of decay rates.

This effect likely has a stronger influence than the presence of multiple obstacles. The additional obstacles provide the NN with more CBF constraints, each associated with its own decay rate, thereby giving the network greater freedom to adjust the trajectory through these outputs. Nonetheless, this influence is smaller compared to that of a longer prediction horizon. Both a longer horizon and multiple obstacles increase the number of constraints and the

overall problem complexity. While a longer horizon introduces stronger temporal coupling through the system dynamics, additional obstacles add spatial coupling between multiple CBF constraints. The latter can reduce the feasible space and make optimization more difficult, whereas a longer horizon generally improves foresight and enables more informed parameter tuning.

In addition, as the prediction horizon increases, the MPC terminal cost becomes less dominant, since the cumulative stage cost contributes more to the overall objective. These findings suggest that tuning the CBF alone can ensure safety while also improving performance.

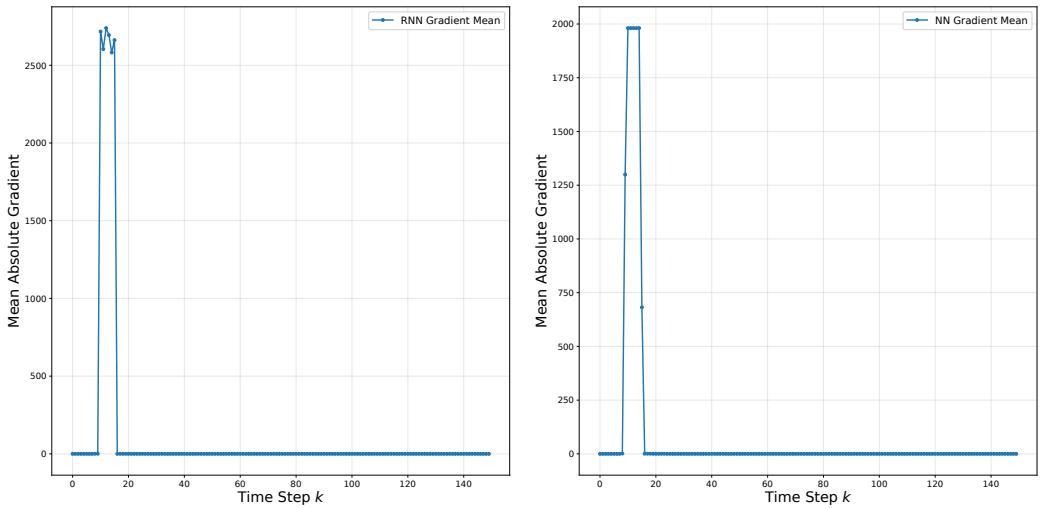
A limitation of both learning methods is that gradient updates with respect to the neural network parameters occur only when the CBF constraint is active. When the constraint is inactive, the corresponding dual variable in the Lagrangian of $Q_\theta(s,a)$, whose role in the MPC-based RL framework was explained in Section 2-2-3, is zero. As a result, the derivative of the Lagrangian with respect to the neural network parameters is naturally also zero. To illustrate this more clearly, consider the Lagrangian of the NN-CBF in (4-7) given below, but now expressed as $Q_\theta(s,a)$ with the added constraint $u_0 = a$.

$$\begin{aligned}\mathcal{L}_\theta(s,a,y^\star) = \ell_{f,\theta}(x_N) + \sum_{k=0}^{N-1} \ell_\theta(x_k,\, u_k) + w_{\text{MPC}} \sum_{k=0}^{N-1}\sum_{i=1}^{\mathcal{O}} \sigma_{k,i} \\ + \cdots + \sum_{k=0}^{N-1}\sum_{i=1}^{\mathcal{O}} \lambda_{k,i}\Big(-h_i(x_{k+1}) + (1-\text{NN}_\theta(\cdot))h_i(x_k) - \sigma_{k,i}\Big) \\ - \sum_{k=0}^{N-1}\sum_{i=1}^{\mathcal{O}} \rho_{k,i}\,\sigma_{k,i} + \zeta^\top(u_0 - a),\end{aligned} \tag{5-9}$$

where $y^\star$ denotes the primal-dual variables and $\lambda_{k,i}, \rho_{k,i}, \zeta$ represent some of the dual variables corresponding to the constraints shown in the Lagrangian above. Taking the derivative of the Lagrangian with respect to the neural network parameters gives:

$$\frac{\partial \mathcal{L}_\theta(s,a,y^\star)}{\partial \theta_{\text{NN}}} = \frac{\partial}{\partial \theta_{\text{NN}}} \sum_{k=0}^{N-1}\sum_{i=1}^{\mathcal{O}} \lambda_{k,i}\Big(-h_i(x_{k+1}) + (1-\text{NN}_\theta(\cdot))h_i(x_k) - \sigma_{k,i}\Big). \tag{5-10}$$

It can be seen that if the constraint is not active and therefore $\lambda_{k,i} = 0$, then the derivative is $\frac{\partial \mathcal{L}_\theta(s,a,y^\star)}{\partial \theta_{\text{NN}}} = 0$. The same applies to the RNN formulation. This effect is confirmed in Figure 5-16, which shows the mean NN and RNN gradients across three episodes. Nonzero gradients appear only between $k = 8$ and $k = 16$, precisely when slack variables are active and the CBF constraints are enforced (see Figures 5-11c and 5-14c). Consequently, learning in NN-CBF and RNN-CBF occurs mainly in these intervals, while outside them potentially valuable information for obstacle avoidance is lost.




**(a)** Mean absolute gradient of the RNN parameters

**(b)** Mean absolute gradient of the NN parameters

**Figure 5-16:** Mean absolute gradients of RNN and NN parameters before training across 150 time steps (averaged over 150 episodes). Gradients remain close to zero for most of the episode, except during the interval where CBF constraints become active.

This principle does not apply to the LOPTD-CBF method, since its learnable parameters appear directly in the objective function and not in the CBF constraint, as shown in (4-2). To address the limitations identified for the neural network approaches, an extension to these methods is proposed in Future works 6-2.

It should be noted that this issue also likely arises due to the use of a value-based reinforcement learning method, namely Q-learning, which estimates the action-value function $Q(s, a)$ through temporal-difference updates. The convergence speed and learning stability of value-based methods depend on the magnitude of the experienced returns, since informative (i.e., sufficiently large) value estimates are required to drive effective updates. In contrast, policy-gradient approaches optimize the expected return directly and are therefore potentially less sensitive to the scaling or sparsity of the value signal. However, alternative policy-gradient methods have not been investigated in this work and could represent a promising direction for future research.

Lastly, a major drawback of both neural network-based methods is the substantial nonlinearities they introduce into the interior-point solver. Although IPOPT is designed to handle nonlinear programs, the optimization becomes increasingly challenging as the network size grows and the MPC horizon lengthens.

# Chapter 6

# Conclusions and Future Work

## 6-1 Conclusion

In this thesis, novel methods for merging MPC, RL and CBFs were proposed. The central idea was to use parameterized MPC with a CBF constraint as a function approximator to learn a safe and higher-performing policy. The proposed methods expanded on this idea by exploring different ways of parameterizing and adapting the class $\mathcal{K}$ function in the CBF condition. The first method, LOPTD-CBF, focused on improving feasibility guarantees while allowing RL to search for better policies. This was achieved by extending and parameterizing the OPTD framework. The second method, NN-CBF, introduced a richer parametrization through a neural network, enabling more expressive safety conditions and yielding policies with better performance than LOPTD-CBF. Finally, the RNN-CBF extended the NN-CBF framework with a recurrent structure to better handle moving obstacles by accounting for temporal dependencies. Numerical experiments confirmed that both LOPTD-CBF and NN-CBF produce safe and improved policies in a simple obstacle avoidance scenario, with NN-CBF outperforming LOPTD-CBF. In the case of time-varying obstacles, initial findings showed that RNN-CBF is able to find a safe policy faster than NN-CBF, although this result is drawn from a single configuration.

Having summarized the main contributions and finding of this thesis the sub-reserach questions of this thesis are answered below.

1. *Which type of CBF formulation yields the best trade-off between safety and performance in MPC-based RL?*
   The CBF formulations that provide the best trade-off between safety and performance among the methods developed are the neural network based approaches NN-CBF and RNN-CBF. The LOPTD-CBF can adjust the decay rate online during rollout, which improves feasibility and can ensure safer behavior while learning in theory, but its parametrization is more limited. In contrast, the NN-CBF and RNN-CBF offer richer, state dependent parametrizations of the class $\mathcal{K}$ function, allowing them to achieve both

safety and higher performance. If trained long enough in time-varying settings both methods can achieve similair performance, however the RNN-CBF can achieve it faster based on preliminary results. Lastly, the choice of CBF formulation that provides the best trade-off between safety and performance also depends on the underlying problem. Other CBF formulations not covered in this work may be more suitable. For example, in cases with stochastic dynamics or uncertainty in the model and obstacles, robust or stochastic CBFs would be required, since the methods developed in this thesis would not be sufficient. This is discussed further in Section 6-2.

2. *How does tuning the CBF condition affect the balance between safety and performance?* To address this question, experiments were conducted in two setups. In one setup both the terminal cost and the CBF condition were parameterized. In the other setup only the CBF condition was parameterized in order to isolate its effect on safety and performance. The numerical results indicate that parameterizing only the CBF condition can improve performance while still guaranteeing safety, provided the prediction horizon is sufficiently long. Learning solely the CBF parameters with a horizon of just one step was not enough to significantly alter the trajectory and the terminal cost played a more prominent role in improving performance, as seen in the static obstacle experiments in 5-1. In contrast, in the dynamic obstacle experiments in 5-2, tuning only the CBF condition already yielded noticeable improvements. A potential reason is that with a longer horizon, the number of CBF constraints increases, giving the neural network more class-$\mathcal{K}$ functions to tune and thereby more degrees of freedom to shape the trajectory. At the same time, the influence of the terminal cost diminishes, since with a longer horizon the stage cost is accumulated over more steps. Therefore, with a sufficiently long horizon, reinforcement learning can improve closed-loop performance by tuning only the CBF condition.

3. *How can MPC-based RL using CBFs guarantee safety in dynamic environments, e.g., multiple dynamic obstacles?*
MPC-based RL can be extended to handle multiple obstacles by assigning each obstacle its own CBF constraint with a parameterized class $\mathcal{K}$ function. Reinforcement learning can then tune the class $\mathcal{K}$ function of each obstacle separately, enabling more nuanced safety handling. This becomes increasingly important as the number of obstacles grows, since the feasible and control invariant sets shrink. For dynamic obstacles, the extension remains straightforward because the discrete CBF condition only requires forward differences. In contrast, a continuous-time CBF condition would require additional derivative terms when computing $\dot{h}(x, u)$. Finally, the proposed RNN-CBF can also account for time-varying obstacles by storing temporal information in its hidden state, allowing it to anticipate obstacle motion and preserve safety.

Finally, addressing the overarching research question: **To what extent can performance be maximized while still guaranteeing safety in MPC-based Reinforcement Learning using Control Barrier Functions?** The findings of this thesis demonstrate that MPC-based RL with a parameterized CBF condition can achieve substantial performance improvements while preserving safety. The LOPTD-CBF shows that feasibility can be maintained even with limited horizons, although performance gains remain moderate. The NN-CBF offers a richer parameterization that yields more effective policies, while the RNN-CBF extends this

capability to dynamic settings and ensures faster convergence. Taken together, these results confirm that RL can substantially enhance performance in safety-critical environments without compromising safety, provided that the MPC and the CBF condition are parameterized appropriately and sufficiently within the MPC-based RL framework.

## 6-2 Future Work

Future research can build on the ideas explored in this thesis by addressing a variety of challenges and extending the framework to different settings. The directions outlined below highlight promising opportunities for improvement and further investigation.

### 6-2-1 Improving the proposed NN-based methods.

The results from Section 5-2 show that the RNN and NN are able to find safe trajectories, but they break the formal safety certification of control invariance by using slack variables. Furthermore, these methods do not update the neural network parameters unless the CBF constraint is active, as explained in 5-2-3.

To tackle both issues, an additional decision variable $z_{k,i}$ can be introduced and incorporated into the CBF constraint as $h_i(x_{k+1}) - h_i(x_k) + \alpha_i(h_i(x_k)) \geq z_{k,i}$. This variable also appears in the MPC cost as a negative term $-z_{k,i}$ scaled by a weight. As a result, since $z_{k,i}$ appears with a negative weight in the cost, the MPC seeks to maximize its value in order to minimize the total objective. At the same time, raising $z_{k,i}$ makes the CBF constraint more conservative by shifting its lower bound upward. This allows the MPC to take more cautious actions in advance, thereby reducing the likelihood of CBF violations and minimizing reliance on slack variables later on. Furthermore, the inclusion of $z_{k,i}$ tightens the CBF constraint and promotes it to remain active. Consequently, when the Lagrangian is constructed for parameter updates, as described in Section 2-2-3, the dual variables of the CBF constraints are more likely to be nonzero, unlike the case observed in Section 5-2. This enables more frequent updates of the neural network parameters since the CBF constraint becomes active more often. The decision variable can then be incorporated into the MPC as follows

$$V_\theta(s) = \min_{X,U,S,\Sigma} \quad \ell_{f,\theta}(x_N) + \sum_{k=0}^{N-1} \ell_\theta(x_k, u_k) + w_{\text{MPC}} \sum_{k=0}^{N-1} \sum_{i=1}^{\mathcal{O}} \sigma_{k,i} - \|x_0\|_2 \sum_{k=0}^{N-1} d^k \sum_{i=1}^{\mathcal{O}} \beta z_{k,i} \tag{6-1a}$$

$$\text{s.t.} \quad x_{k+1} = f(x_k, u_k), \qquad k = 0, \dots, N-1, \tag{6-1b}$$

$$x_k \in \mathcal{S}, \qquad k = 0, \dots, N, \tag{6-1c}$$

$$u_k \in \mathcal{A}, \qquad k = 0, \dots, N-1, \tag{6-1d}$$

$$x_0 = s, \tag{6-1e}$$

$$h(x_{k+1}) - (1 - \text{NN}_\theta(\cdot)) \geq -\sigma_{k,i} + z_{k,i}, \quad \begin{matrix} k = 0, \dots, N-1, \\ i = 0, \dots, \mathcal{O}, \end{matrix} \tag{6-1f}$$

$$\Sigma_{k,i} \geq 0, \quad \begin{matrix} k = 0, \dots, N-1, \\ i = 0, \dots, \mathcal{O}, \end{matrix} \tag{6-1g}$$

where $d \in (0, 1)$ is a decay term and $\beta \in \mathbb{R}< 0$ is a scalar multiplier in the appended cost for the $z_{k,i}$ decision variable. The new objective term is designed to decrease as the system approaches the target. Without this decrease, as the system moves closer to the origin, the contribution of the $z_{k,i}$ term would become dominant in the cost. Very close to the origin, the stage and terminal costs $x^\top Px$ and $x^\top Qx$ are small, so if the $z_{k,i}$ cost does not decay with the state, it could outweigh these terms and prevent convergence to the origin. For this reason the term is scaled with the state. An initial idea was to include a multiplication between the 2-norm of the predicted states and the decision variable $z_{k,i}$, expressed as $-\sum_{k=0}^{N-1}\sum_{i=1}^{\mathcal{O}} \beta z_{k,i}\|x_k\|_2$. However, this introduces a bilinear term, since it multiplies two decision variables, which makes the problem non-convex and more difficult for the solver. As an alternative only the first state $x_0$ is used, since it represents the current state of the system and is not a decision variable. The decay factor $d$ then serves to approximate the reduction that would naturally occur in the predicted states as they move closer to the solution, leading to the formulation presented above.

However, this method has not yet been explored numerically in detail. Although it enables gradient updates outside the usual instances, the dual variables associated with the CBF constraints where the slack variables $S$ are active are typically much larger than the other dual variables in the Lagrangian. As a result, the gradient updates remain dominated by constraint violations. This limitation requires further consideration.

### 6-2-2 Uncertainties and Disturbances

In real-world applications, systems are inevitably subject to model uncertainties and external disturbances. While these aspects were not explicitly addressed in this thesis, they present a natural direction for further research. To address them, different types of CBFs can be employed. The most prominent methods in the literature are robust and stochastic CBFs [21, 6, 7, 14]. Robust CBFs typically introduce a robustness term into the CBF condition, which serves as a buffer to account for worst-case uncertainties. This robustness term is framed in various ways across different formulations, some of which include tunable parameters. It would be interesting to investigate whether these tunable parameters could be learned alongside the methods in this thesis to account for disturbances or model mismatch. Stochastic CBFs, on the other hand, aim to ensure safety under stochastic uncertainty by employing probabilistic bounds to define and maintain safe regions. The ideas in this thesis of tuning a parametrized class $\mathcal{K}$ function within the MPC-based RL framework could be applied to improve the performance of a stochastic CBF. Work in this direction already exists, as shown in [3]. However, this paper does not employ a neural network-based approach to learn the class $\mathcal{K}$ function, as done in NN-CBF and RNN-CBF.

### 6-2-3 Higher Relative Degree

Another possible extension of the neural methods (NN-CBF and RNN-CBF) presented in this thesis would be to account for systems with higher relative degree, which have not been considered here (see 2-3-3 for a discussion of relative degree). This could be incorporated into the NN-CBF and RNN-CBF frameworks. In this setting, discrete HOCBFs are defined as

$$\psi_0(x_k) = h(x_k), \tag{6-2}$$

$$\psi_1(x_k) = \Delta\psi_0(x_k, u_k) + \alpha_1(\psi_0(x_k)), \tag{6-3}$$

$$\vdots$$

$$\psi_r(x_k) = \Delta\psi_{r-1}(x_k, u_k) + \alpha_r(\psi_{r-1}(x_k)), \tag{6-4}$$

where $\Delta\psi(x_k, u_k) = \psi(x_{k+1}) - \psi(x_k)$. This approach could extend this idea by replacing the class $\mathcal{K}$ functions with learnable discrete eCBF parameters:

$$\psi_0(x_k) = h(x_k), \tag{6-5}$$

$$\psi_1(x_k) = \Delta\psi_0(x_k, u_k) + \gamma_1\psi_0(x_k), \tag{6-6}$$

$$\vdots$$

$$\psi_r(x_k) = \Delta\psi_{r-1}(x_k, u_k) + \gamma_r\psi_{r-1}(x_k), \tag{6-7}$$

with the decay rates $\gamma_i$ provided as outputs of the NN or RNN. Exploring the effectiveness of this extension on higher-order systems would be valuable, since the added outputs per CBF condition increase training complexity. It would also be interesting to compare the results to the method in [26], which addresses continuous HOCBFs in a related way.

### 6-2-4 Comparisons with Other Methods

Even though the methods presented in this thesis show promising results, no direct comparison with [26] is included. Such an evaluation would be important to confirm the advantages of learning in this framework and to determine whether the proposed method offers a general improvement over [26]. Moreover, implementing the proposed method alongside [13] and [34] on an appropriate example is also of interest. This was not pursued for the numerical example in Chapter 5 because using a continuous-time CBF for the double integrator would require a CBF of relative degree two, whereas the discrete CBFs used in this thesis have relative degree one.

### 6-2-5 Unknown CBF

When obstacles are not known in advance or the CBF condition is difficult to specify, learning the CBF itself may be beneficial. Building on Section 3-2-1, one possible direction is to parameterize the CBF with a neural network and learn it jointly with the MPC, which once again would serve as the function approximator for RL. However, this introduces challenges, since learning an unknown CBF is likely to cause safety violations during training. Initialization of the CBF parameters is then critical. A conservative initialization can make the system overly cautious and weaken exploration, while a less conservative initialization can cause frequent violations.

### 6-2-6 Complex Numerical Experiment

So far, the frameworks in the thesis have been implemented on a double integrator system and extending it to multiple dynamic obstacles has already introduced additional complexity. A natural next step would be to test how the method performs on non-linear and more complex systems. Such an extension would raise computational challenges and test the learning capability of the approach in more complex scenarios that are representative of real-life problems.

### 6-2-7 Learning a generalizable class $\mathcal{K}$ function

The NN-CBF and RNN-CBF methods considered in this thesis rely on neural networks. This raises the possibility of learning more general class $\mathcal{K}$ functions that guarantee safety with respect to the system itself rather than being tailored to specific scenarios. The objective would be to extend safety guarantees to unseen situations, including cases that have not been encountered during training. In practice, this would require exposing the network to a variety of scenarios during training so that it can capture more general features and account for obstacles that were not explicitly experienced before. To support this generalization, the network inputs would need to be adapted. For instance, dependencies on system inputs and velocities could be preserved, while direct dependence on position might be avoided since obstacles may appear in varying locations.

Handling multiple obstacles also becomes an important consideration. A larger number of obstacles implies more outputs and a correspondingly larger network. However, if the aim is to maintain generality, it may not be feasible to continuously expand the input and output dimensions of the network after it has been trained. One practical solution is to design the network and the MPC to account for a fixed number $\mathcal{O}$ of obstacles, where only the closest obstacles are considered. This approach, however, introduces a subjective choice of $\mathcal{O}$ and may fail to capture situations where an obstacle slightly farther away still poses a significant safety risk.

# Appendix A

# Settings of the Experiments

This Appendix chapter presents all hyperparameters used for training in the experiments of this thesis, as well as the bounds for the learnable parameters.

# A-1 LOPTD-CBF Static Obstacle Experiment

**Table A-1:** Experiment Settings

| Name | Value | Description |
|---|---|---|
| Episode Duration | 3000 | Time steps per episode. |
| Number of Episodes | 3000 | Episodes used for the reported run. |
| Sampling Time | 0.2 s | Discrete-time step $\Delta t$. |
| Patience Threshold | 10 | Episodes without improvement before LR decay. |
| $P_\omega$ | 1000 | Weighting parameter controlling how strongly $\omega$ is driven toward $\bar{\omega}$. |
| $\bar{\omega}$ | 0.4 | Reference decay rate toward which $\omega$ is regulated. |
| Learning Rate Decay Factor | 0.1 | Multiplier applied when patience is exceeded. |
| Slack Penalty (MPC) | 20,000,000.0 | Cost weight on slacks in MPC objective. |
| Slack Penalty (RL) | 1,000.0 | Cost weight on slacks in RL stage cost. |
| Adam $\beta_1, \beta_2, \epsilon$ | 0.9, 0.999, $10^{-8}$ | ADAM parameters |
| Seed | 69 | Random seed for reproducibility. |
| Alpha (Learning Rate) | 0.5 | Base step size for parameter updates. |
| Gamma RL $\gamma_{\text{RL}}$ | 0.95 | RL Discount factor. |
| Initial Noise Scale | 4 | Exploration noise scaling at start. |
| Noise Variance | 5 | Variance of exploration noise. |
| Decay Rate of Noise | 0.02276277904418933 | Per-update decay factor for noise scale. |
| Noise Scale at End | 0.001 | Target exploration scale at end of training. |
| Replay Buffer | 30000 | Number of recent samples used for updates. |
| Episode Update Frequency | 10 | Perform updates every 10 episodes. |
| Per-parameter LR vector | $[\alpha, \alpha, \alpha, \alpha, \alpha, 0.001\alpha]$ | Full $\alpha$ on $P$ diag; Full $\alpha$ on $P_\omega$; $\alpha$*0.001 on $\bar{\omega}$ . |

**Table A-2:** LOPDT-CBF Learnable Parameter Bounds

| Learnable RL Parameter | $\boldsymbol{\theta}_{\text{lb}}$ | $\boldsymbol{\theta}_{\text{ub}}$ | $\Delta\boldsymbol{\theta}_{\text{lb}}$ | $\Delta\boldsymbol{\theta}_{\text{ub}}$ |
|---|---|---|---|---|
| $P$ | 0 | $\infty$ | $-\infty$ | $\infty$ |
| $P_\omega$ | 0 | $\infty$ | $-\infty$ | $\infty$ |
| $\bar{\omega}$ | $10^{-6}$ | 1 | $-\infty$ | $\infty$ |

**Table A-3:** Obstacle & Environment Configuration

| ID | Mode | Bounds (for mode) | Speed | Initial Direction | Position / Radius |
|---|---|---|---|---|---|
| 1 | `static` | - | 0.0 | - | $(-2.0, -2.25)$ / 1.5 |

## A-2 NN-CBF Static Obstacle Experiment

**Table A-4:** Experiment Settings

| Name | Value | Description |
|---|---|---|
| Episode Duration | 3000 | Time steps per episode. |
| Number of Episodes | 3000 | Episodes used for the reported run. |
| Sampling Time | 0.2 s | Discrete-time step $\Delta t$. |
| Layers List | [5, 8, 8, 8, 1] | RNN-CBF architecture: input, hidden, hidden, output. |
| Patience Threshold | 10 | Episodes without improvement before LR decay. |
| Learning Rate Decay Factor | 0.1 | Multiplier applied when patience is exceeded. |
| Slack Penalty (MPC) | 20,000,000.0 | Cost weight on slacks in MPC objective. |
| Slack Penalty (RL) | 1,000.0 | Cost weight on slacks in RL stage cost. |
| Adam $\beta_1, \beta_2, \epsilon$ | 0.9, 0.999, $10^{-8}$ | ADAM parameters |
| Seed | 69 | Random seed for reproducibility. |
| Alpha (Learning Rate) | 0.5 | Base step size for parameter updates. |
| Gamma RL $\gamma_{\mathrm{RL}}$ | 0.95 | RL Discount factor. |
| Initial Noise Scale | 4 | Exploration noise scaling at start. |
| Noise Variance | 5 | Variance of exploration noise. |
| Decay Rate of Noise | 0.02276277904418933 | Per-update decay factor for noise scale. |
| Noise Scale at End | 0.001 | Target exploration scale at end of training. |
| Replay Buffer | 30000 | Number of recent samples used for updates. |
| Episode Update Frequency | 10 | Perform updates every 10 episodes. |
| Per-parameter LR vector | $[\alpha, \alpha, \alpha, \alpha,\ 0.01\alpha, \ldots]$ | Full $\alpha$ on $P$ diag; $0.01\alpha$ for NN params. |

**Table A-5:** NN-CBF Learnable Parameter Bounds

| Learnable RL Parameter | $\boldsymbol{\theta}_{\mathrm{lb}}$ | $\boldsymbol{\theta}_{\mathrm{ub}}$ | $\Delta\boldsymbol{\theta}_{\mathrm{lb}}$ | $\Delta\boldsymbol{\theta}_{\mathrm{ub}}$ |
|---|---|---|---|---|
| $P$ | 0 | $\infty$ | $-\infty$ | $\infty$ |
| NN parameters | $-\infty$ | $\infty$ | $-\infty$ | $\infty$ |

**Table A-6:** Obstacle & Environment Configuration

| ID | Mode | Bounds (for mode) | Speed | Initial Direction | Position / Radius |
|---|---|---|---|---|---|
| 1 | `static` | - | 0.0 | - | $(-2.0, -2.25)$ / 1.5 |

## A-3 NN-CBF Dynamic Obstacles Experiment

**Table A-7:** Experiment Settings

| Name | Value | Description |
|---|---|---|
| Episode Duration | 150 | Time steps per episode. |
| Number of Episodes | 3000 | Episodes used for the reported run. |
| Sampling Time | 0.2 s | Discrete-time step $\Delta t$. |
| Layers List | [13, 16, 16, 16, 3] | RNN-CBF architecture: input, hidden, hidden, output. |
| Patience Threshold | 20 | Episodes without improvement before LR decay. |
| Learning Rate Decay Factor | 0.1 | Multiplier applied when patience is exceeded. |
| Slack Penalty (MPC) | 20,000,000.0 | Cost weight on slacks in MPC objective. |
| Slack Penalty (RL) | 40,000.0 | Cost weight on slacks in RL stage cost. |
| Adam $\beta_1, \beta_2, \epsilon$ | 0.9, 0.999, $10^{-8}$ | ADAM parameters |
| Seed | 69 | Random seed for reproducibility. |
| Alpha (Learning Rate) | 0.007 | Base step size for parameter updates. |
| Gamma RL $\gamma_{\mathrm{RL}}$ | 0.95 | RL Discount factor. |
| Initial Noise Scale | 10 | Exploration noise scaling at start. |
| Noise Variance | 5 | Variance of exploration noise. |
| Decay Rate of Noise | 0.535841116638722 | Per-update decay factor for noise scale. |
| Noise Scale at End | 0.01 | Target exploration scale at end of training. |
| Replay Buffer | 750 | Number of recent samples used for updates. |
| Episode Update Frequency | 5 | Perform updates every 5 episodes. |
| Per-parameter LR vector | $[\alpha, \alpha, \alpha, \alpha,\ 0.01\alpha, \ldots]$ | Full $\alpha$ on $P$ diag; $0.01\alpha$ for NN params. |

**Table A-8:** NN-CBF Learnable Parameter Bounds

| Learnable RL Parameter | $\boldsymbol{\theta}_{\mathrm{lb}}$ | $\boldsymbol{\theta}_{\mathrm{ub}}$ | $\Delta\boldsymbol{\theta}_{\mathrm{lb}}$ | $\Delta\boldsymbol{\theta}_{\mathrm{ub}}$ |
|---|---|---|---|---|
| NN parameters | $-\infty$ | $\infty$ | $-\infty$ | $\infty$ |

**Table A-9:** Obstacle & Environment Configuration

| ID | Mode | Bounds (for mode) | Speed | Initial Direction | Position / Radius |
|---|---|---|---|---|---|
| 1 | `step_bounce` | $x \in (-4.0, 0.0)$ | 2.3 | $+1$ | $(-2.0, -1.5)$ / 0.7 |
| 2 | `step_bounce` | $x \in (-4.0, 1.0)$ | 2.0 | $-1$ | $(-3.0, -3.3)$ / 0.7 |
| 3 | `static` | - | 0.0 | - | $(-2.0, 0.0)$ / 1.0 |

## A-4 RNN-CBF Dynamic Obstacles Experiment

**Table A-10:** Experiment Settings

| Name | Value | Description |
|---|---|---|
| Episode Duration | 150 | Time steps per episode. |
| Number of Episodes | 3000 | Episodes used for the reported run. |
| Sampling Time | 0.2 s | Discrete-time step $\Delta t$. |
| Layers List | [13, 16, 16, 16, 3] | RNN-CBF architecture: input, hidden, hidden, output. |
| Patience Threshold | 20 | Episodes without improvement before LR decay. |
| Learning Rate Decay Factor | 0.1 | Multiplier applied when patience is exceeded. |
| Slack Penalty (MPC) | 20,000,000.0 | Cost weight on slacks in MPC objective. |
| Slack Penalty (RL) | 40,000.0 | Cost weight on slacks in RL stage cost. |
| Recurrent init | Spectral radius target $\approx$ 0.95 | Recurrent $W_{hh}$ scaled to $\rho(W) \approx 0.95$. |
| Adam $\beta_1, \beta_2, \epsilon$ | 0.9, 0.999, $10^{-8}$ | ADAM parameters |
| Seed | 69 | Random seed for reproducibility. |
| Alpha (Learning Rate) | 0.007 | Base step size for parameter updates. |
| Gamma RL $\gamma_{\mathrm{RL}}$ | 0.95 | RL Discount factor. |
| Initial Noise Scale | 10 | Exploration noise scaling at start. |
| Noise Variance | 5 | Variance of exploration noise. |
| Decay Rate of Noise | 0.535841116638722 | Per-update decay factor for noise scale. |
| Noise Scale at End | 0.01 | Target exploration scale at end of training. |
| Replay Buffer | 750 | Number of recent samples used for updates. |
| Episode Update Freq. | 5 | Perform updates every 5 episodes. |
| Per-parameter LR vector | $[\alpha, \alpha, \alpha, \alpha,\ 0.01\alpha, \ldots]$ | Full $\alpha$ on $P$ diag; $0.01\alpha$ for RNN params. |
| RNN wamrup steps | 15 | Initial 15 steps allow the hidden state to converge to a stable value. |

**Table A-11:** RNN-CBF Learnable Parameter Bounds

| Learnable RL Parameter | $\boldsymbol{\theta}_{\mathrm{lb}}$ | $\boldsymbol{\theta}_{\mathrm{ub}}$ | $\Delta\boldsymbol{\theta}_{\mathrm{lb}}$ | $\Delta\boldsymbol{\theta}_{\mathrm{ub}}$ |
|---|---|---|---|---|
| RNN parameters | $-\infty$ | $\infty$ | $-\infty$ | $\infty$ |

**Table A-12:** Obstacle & Environment Configuration

| ID | Mode | Bounds (for mode) | Speed | Initial Direction | Position / Radius |
|---|---|---|---|---|---|
| 1 | `step_bounce` | $x \in (-4.0, 0.0)$ | 2.3 | $+1$ | $(-2.0, -1.5)$ / 0.7 |
| 2 | `step_bounce` | $x \in (-4.0, 1.0)$ | 2.0 | $-1$ | $(-3.0, -3.3)$ / 0.7 |
| 3 | `static` | - | 0.0 | - | $(-2.0, 0.0)$ / 1.0 |

# Appendix B

# Adam

Adaptive Moment Estimation also known as Adam, replaces a single global learning rate with per-parameter adaptation by tracking two exponential moving averages of the gradients [20]. The first moment $m_t$ estimates the mean direction of recent gradients and the second moment $v_t$ estimates their uncentered variance. For parameters $\theta \in \mathbb{R}^d$ and gradient $g_t = \nabla_\theta \mathcal{L}_\theta(s, y^\star)$ at iteration $t \geq 1$, the moments are updated as

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2)\, g_t \odot g_t,$$

with $m_0 = v_0 = 0$ and decay rates $\beta_1, \beta_2 \in (0, 1)$. Because these moving averages are initialized at zero, they tend to be biased towards zero, most notably during the first few steps. To correct for this bias, Adam computes bias-corrected terms of the previous estimates.

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}.$$

The resulting parameter update is then

$$\Delta\theta_t = -\,\alpha_t \,\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}, \qquad \theta_{t+1} = \theta_t + \Delta\theta_t,$$

where $\alpha_t > 0$ denotes a (possibly elementwise) learning rate, $\varepsilon > 0$ is a small numerical constant. An equivalent implementation is shown by combining all the terms into one step

$$\Delta\theta_t = -\,\frac{\alpha_t}{1-\beta_1^t}\,\frac{m_t}{\sqrt{v_t}/\sqrt{1-\beta_2^t} + \varepsilon},$$

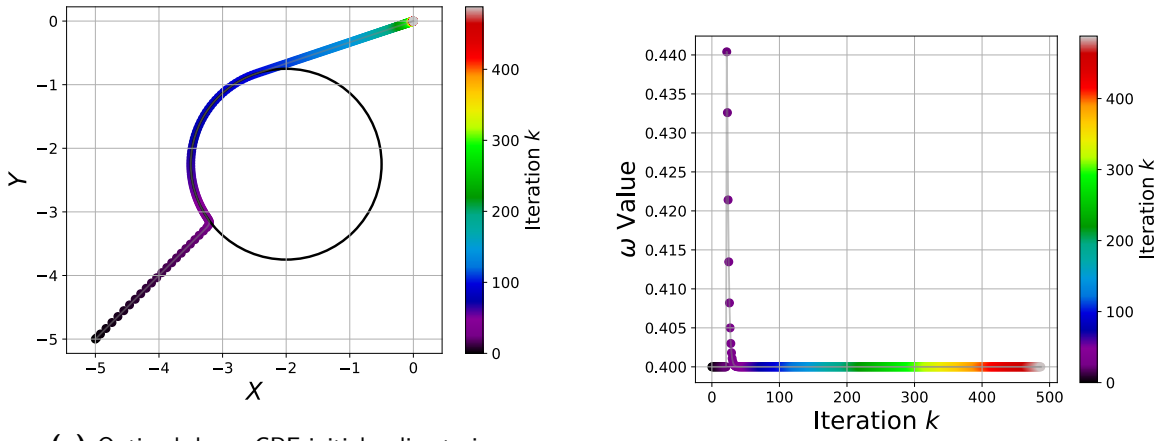
which is algebraically identical to the standard Adam update above.

# Appendix C

# Static Obstacle Experiment Without Terminal Cost Parametrization

This Appendix chapter shows that the RL was unable to learn a good policy when the terminal cost was not parametrized in the static example. This suggests that having a parameterization only over the class $\mathcal{K}$ function and not the terminal cost is likely insufficient to capture the optimal value function, and thus also the optimal policy.

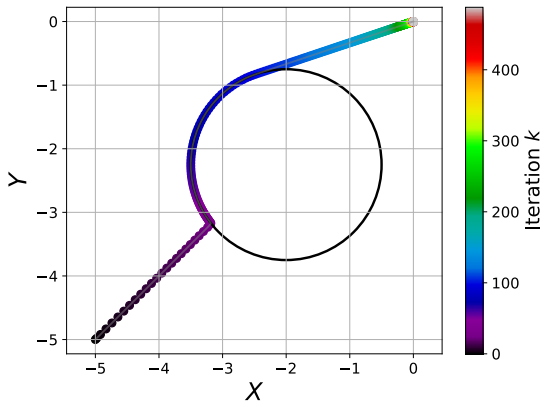
## C-0-1 LOPTD-CBF



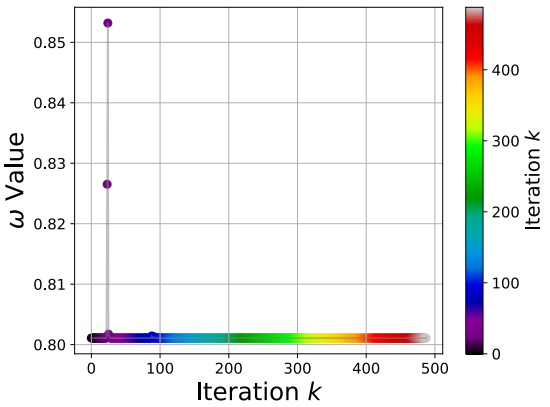

**(a)** Optimal decay CBF initial policy trajectory

**(b)** Optimal decay CBF initial policy $\omega$ plot

**Figure C-1:** Policy before training

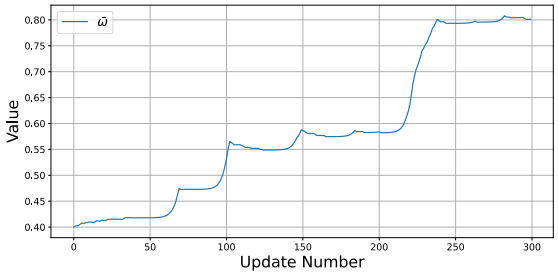



**(a)** Optimal decay CBF final policy trajectory

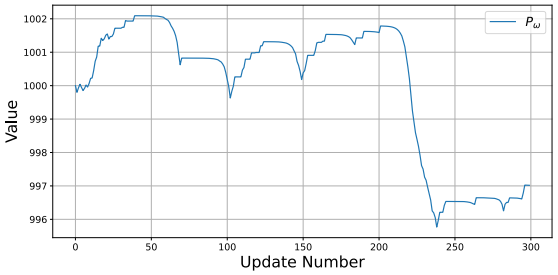



**(b)** Optimal decay CBF final policy $\omega$ plot

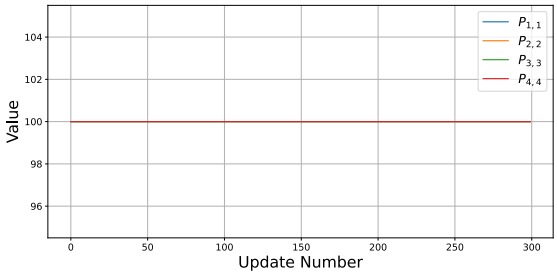
**Figure C-2:** Policy after training




**(a)** Evolution of $\bar{\omega}$ during training




**(b)** Evolution of $P_\omega$ during training




**(c)** Evolution of terminal cost matrix $P$ during training

**Figure C-3:** Evolution of learned parameters during training

## C-0-2 NN-CBF

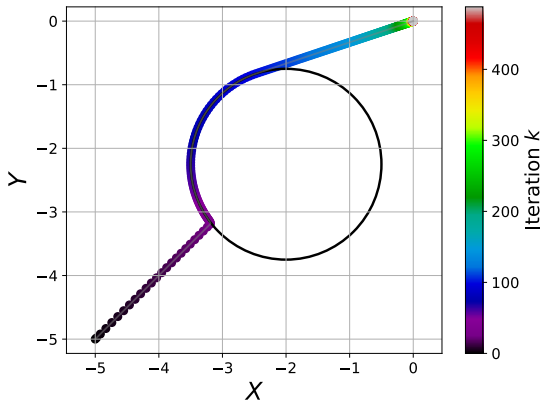
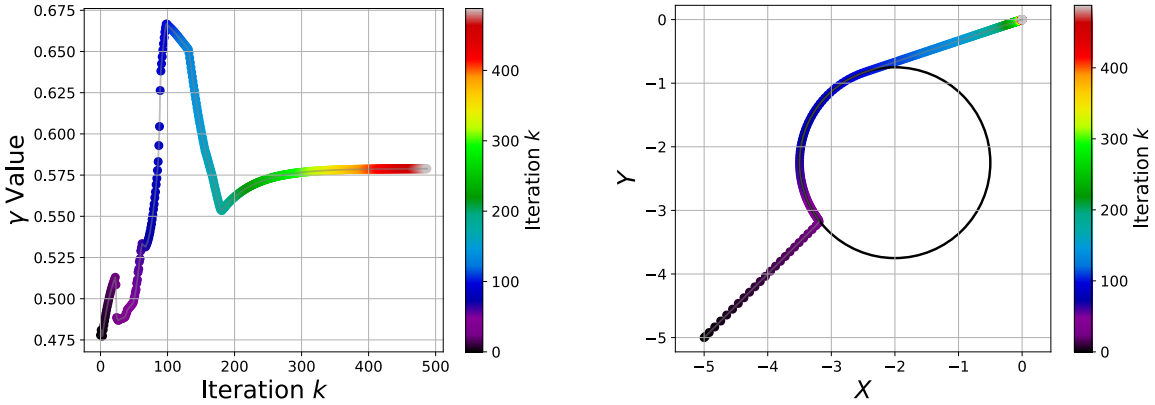



**Figure C-4:** NN-CBF initial policy trajectory




**(a)** NN-CBF final policy trajectory

**(b)** NN-CBF final policy $\gamma$ plot

**Figure C-5:** Policy after training

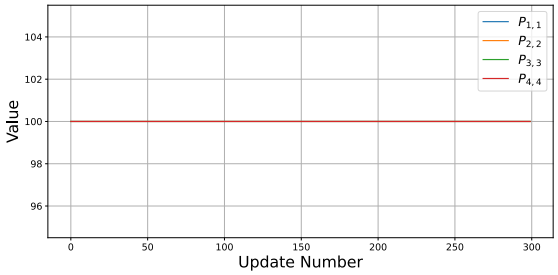



**Figure C-6:** Evolution of terminal cost matrix $P$ during training

# Appendix D

# RNN-CBF Alternative Solution

Further training of the RNN-CBF without slowing down the learning earlier yields the solution shown in Figure D-1, achieving a stage cost of **6734** with a policy that does not use slacks.

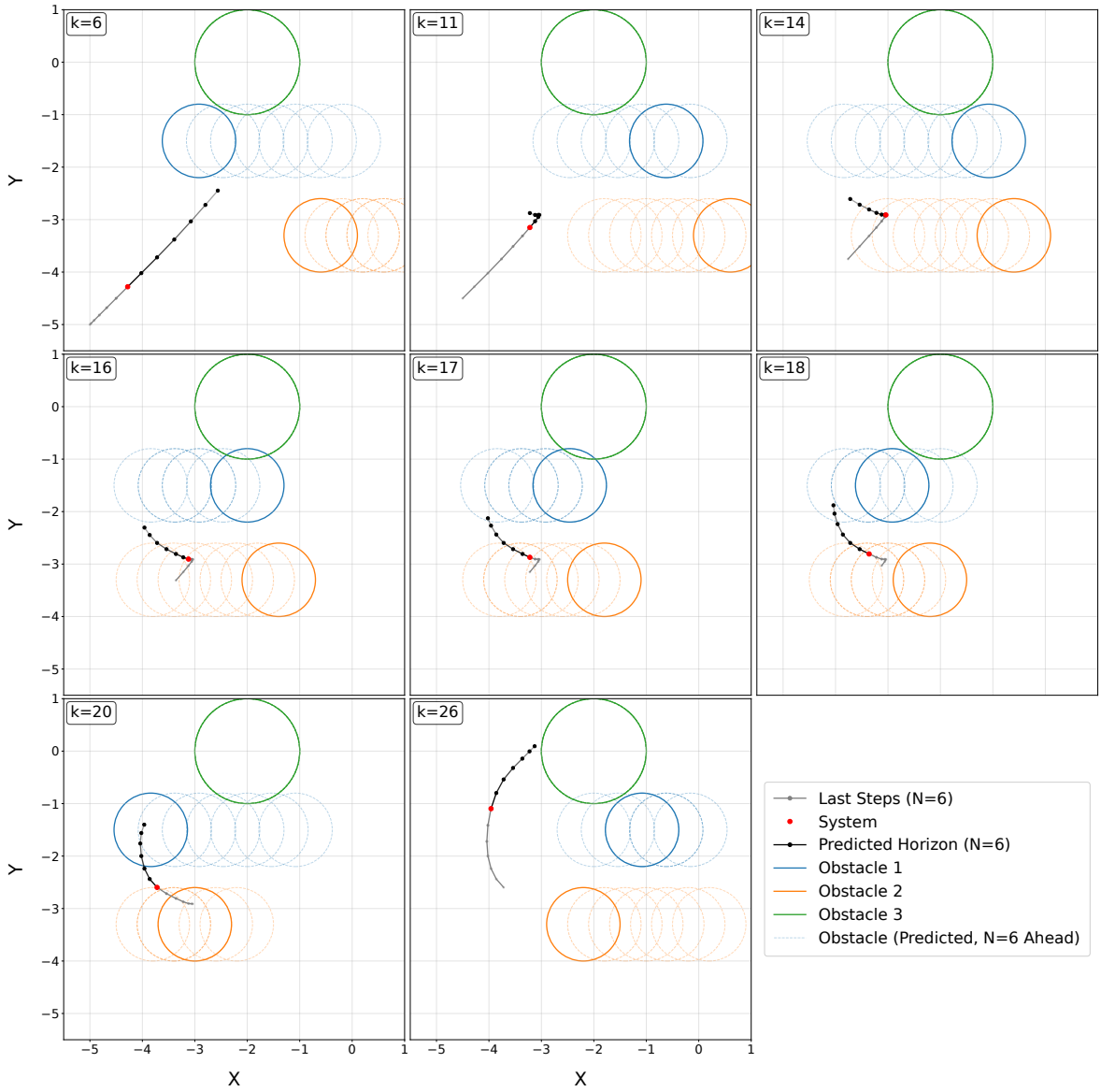



**Figure D-1:** Snapshots of the alternative RNN-CBF policy after training, obtained without slacks. The red dot marks the current system position, the black trail shows the predicted horizon and the grey line shows the last six steps of the system trajectory. The solid circles indicate the current positions of the static (green) and dynamic (blue and orange) obstacles, while the dotted circles indicate their predicted positions six time steps ahead.

# Appendix E

# Paper Draft

A draft of a paper summarizing the findings of the thesis is given below.

# Safe model-based Reinforcement Learning via Model Predictive Control and Control Barrier Functions

Kerim Dzhumageldyev, Fillipo Airaldi, Azita Dabiri

***Abstract*—Optimal control strategies are often combined with safety certificates to ensure both performance and safety in safety-critical systems. A prominent example is combining Model Predictive Control (MPC) with Control Barrier Functions (CBF). Yet, tuning MPC parameters and choosing an appropriate class $\mathcal{K}$ function in the CBF is challenging and problem dependent. This paper introduces a safe model-based Reinforcement Learning (RL) framework where a parameterized MPC incorporates a CBF with a parameterized class $\mathcal{K}$ function and serves as a function approximator to learn improved safe control policies. Three variations are introduced, distinguished by the way the class $\mathcal{K}$ function is parameterized. The Learnable Optimal Decay CBF extends the Optimal Decay CBF by allowing RL to tune the optimal decay parameters, enhancing performance while preserving feasibility and safety guarantees. The Neural Network CBF parametrizes the decay term of a discrete exponential CBF with a neural network, enabling richer state-dependent safety conditions. Finally, the Recurrent Neural Network CBF extends the Neural Network CBF with a recurrent architecture to handle time-varying CBF constraints. Numerical experiments on a discrete double-integrator with static and dynamic obstacles demonstrate that the proposed methods improve performance while ensuring safety, each offering distinct trade-offs in performance, feasibility and complexity.**

## I. Introduction

Safety is a concern at the core of numerous problems in control. Over the years, various methodologies have been developed to ensure safety in control systems. Among these, Control Barrier Functions (CBFs) have become one of the most prominent tools in the literature. In particular, they enforce safety by ensuring that system trajectories remain within a prescribed safe set [1], [2]. More recently, CBFs have been combined with Model Predictive Control (MPC), which provides a predictive framework for optimizing control performance, while CBFs guarantee forward invariance of a safe set over the horizon.

The effectiveness of this integration, however, hinges on a handful of design choices. Among others, weights in the MPC cost, the prediction horizon and the class $\mathcal{K}$ function in the CBF all shape the resulting controller. Poorly chosen values can shrink the feasible set, reduce performance or even render the problem infeasible. Notably, the choice of the class $\mathcal{K}$ function is particularly delicate, as it directly affects how conservative the safety constraint is and creates a trade-off between safety and performance.

To address the challenge of selecting a suitable class $\mathcal{K}$ function, several learning-based approaches have been proposed. One line of work models a generic class $\mathcal{K}$ function with a Neural Network (NN) trained jointly with a Reinforcement Learning (RL) policy [3]. A Quadratic Program (QP) formulation with a CBF constraint incorporates this learned function, ensuring safety, while RL training alleviates the inherent myopic limitations of the QP. While QP-based methods enforce safety at each step and RL training helps curb myopic behavior, they do not propagate safety information or plan trajectories across the horizon, which can lead to infeasibility during training before feasible RL policies are found. By contrast, MPC enforces safety by optimizing over the entire horizon rather than step-by-step, providing foresight that preserves feasibility more reliably during training [4]. Lastly, since the class $\mathcal{K}$ function is fixed during training, poor initialization can still lead to infeasibility.

Another approach introduces a learnable penalty term that scales a user-specified class $\mathcal{K}$ function in the CBF condition [5]. Since the class $\mathcal{K}$ function itself is fixed, its choice remains arbitrary. In this approach, learning is performed through a differentiable QP with supervised training, which means performance is dependent on the quality of the nominal controller used for data generation.

A third direction combines MPC–CBF with deep RL, where the RL policy outputs the parametrization for both the MPC controller and CBF constraints [6]. This avoids differentiating through the optimization problem and reduces computation, but gradients are not propagated through MPC, making the method sample-inefficient. Safety and feasibility are also not guaranteed, as CBF parameters update across episodes rather than within rollouts.

Though the present work focuses on deterministic formulations, there exist other recent works that integrate probabilistic or uncertainty-aware formulations into MPC–CBF–RL frameworks, either for stochastic systems [7] or online parameter adaptation [8].

As an alternative, we propose using MPC as the function approximator instead of a NN as in [6], and learning a parametrized class $\mathcal{K}$ function within the CBF constraint. The advantage of leveraging MPC as approximation instead of an NN is that it can directly incorporate dynamics, input, and state constraints, while retaining guarantees such as stability and recursive feasibility. Compared to [6], our approach is thus more sample efficient, since gradients are propagated through the MPC optimization problem rather than via black-box policy updates. Furthermore, while QP-based methods [3], [5] remain limited by their step-wise nature and inability to

plan coherent safe trajectories multiple steps ahead, the MPC scheme is able to yield policies that are optimal and guarantee safety across a prediction horizon.

The contribution of this paper is to propose three different methods to parametrize the class $\mathcal{K}$ function in the context of MPC as function approximation for RL, each with its own distinct trade-offs:

- **Learnable Optimal Decay CBF (LOPTD-CBF)** extends the optimal-decay framework from [9] by making decay parameters learnable, improving feasibility while maintaining safety guarantees.
- **Neural Network CBF (NN-CBF)** replaces fixed decay rates with a feedforward NN that outputs state-dependent decay functions, removing horizon dependence and enabling richer mappings.
- **Recurrent Neural Network CBF (RNN-CBF)** extends NN-CBF with a recurrent neural network (RNN) architecture that incorporates temporal context, improving performance in environments with time-varying constraints.

The paper is structured as follows. Section II gives an an overview of MPC-based RL, along with discrete CBF, discrete exponetial CBF and optimal decay CBF. This is used to motivate Section III. The effectiveness of these methods is tested in Section IV on two obstacle avoidance tasks with different numbers of obstacles. Lastly, Section V concludes the paper and highlights future research directions.

## II. Background

This chapter details the theoretical background used in the paper.

### A. Control Barrier Functions

Here we review discrete CBFs and their incorporation into MPC, followed by discrete exponential CBF and optimal decay CBF, which form the basis of our approaches.

*1) Discrete Control Barrier Functions:* Consider a discrete-time system

$$s_{k+1} = f(s_k, a_k), \tag{1}$$

where $s_k \in \mathcal{S} \subseteq \mathbb{R}^{n_s}$ is the state and $a_k \in \mathcal{A} \subseteq \mathbb{R}^{n_a}$ is the action. Here, $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces. The safe set is defined as the superlevel set of $h : \mathcal{S} \to \mathbb{R}$ : $C = \{s \in \mathcal{S} \mid h(s) \geq 0\}$. The discrete CBF condition ensures forward invariance of $C$ at time step $k$ by requiring that there exists an input $a_k$ such that

$$h(s_{k+1}) - h(s_k) \geq -\alpha(h(s_k)), \quad s \in C, \tag{2}$$

where $\alpha$ is a class $\mathcal{K}$ function satisfying $\alpha(r) < r \ \forall \ r > 0$ [10].

Embedding this condition into MPC involves adding the CBF constraint at each time step in the MPC horizon, alongside the standard dynamics and input constraints [11]. This ensures that the trajectory remains safe throughout the horizon while optimizing performance.

*2) Discrete Exponential Control Barrier Functions:* Discrete exponential CBFs extend nominal CBFs by selecting the class $\mathcal{K}$ function in (2) as $\alpha(r) = \gamma r$, with $\gamma \in (0, 1]$ controlling the decay rate, which imposes an exponential decay on the safety constraint over time [1], [10], [12].

*3) Optimal Decay CBFs:* A limitation of nominal and exponential CBFs is that they may become infeasible under input constraints, as no admissible action may satisfy the condition. Optimal Decay CBFs (OPTD CBFs) address this by making the decay rate $\gamma$ of the discrete exponential CBF adjustable [9]. The fixed decay rate is replaced by the decision variable $\omega$, giving $h(s_{k+1}) - h(s_k) \geq -\omega h(s_k)$, jointly optimized with the input in the following QP

$$\begin{aligned} \min_{u \in \mathcal{A}, \omega \in (0,1]} \quad & \tfrac{1}{2}|u - k(s_k)|^2 + P_\omega(\omega - \bar{\omega})^2 && \text{(3a)} \\ \text{s.t.} \quad & h(f(s_k, u)) - h(s_k) \geq -\omega h(s_k), && \text{(3b)} \end{aligned}$$

where $\bar{\omega}$ is a reference decay rate, $P_\omega$ is a penalty weight and $k : \mathcal{S} \to \mathcal{A}$ is a nominal controller. Small $P_\omega$ values allow $\omega$ to adapt more freely, while large values keep it close to $\bar{\omega}$. In the limit $P_\omega \to \infty$ with $\bar{\omega} = 1$, the constraint reduces to the nominal CBF [9]. Thus, the practitioner can tune $P_\omega$ and $\bar{\omega}$ to balance adaptability of $\omega$ against adherence to a desired decay rate.

### B. MPC as function approximator in RL

The objective of the RL agent is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$, that minimizes the expected discounted return

$$J(\pi) \coloneqq \sum_{k=0}^{T} \gamma_{RL}^k L(s_k, \pi(s_k)), \tag{4}$$

where $L : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the stage cost, $\gamma_{RL} \in (0, 1]$ is the discount factor and $T$ is the task length [13]. The optimal policy $\pi^\star$ is given then by:

$$\pi^\star = \arg\min_{\pi} J(\pi). \tag{5}$$

In general, the true optimal policy $\pi^\star$ and value functions $V^\star, Q^\star$ are difficult to compute exactly. One solution is to employ function approximation schemes to approximate these. Among others, such approximation scheme can be delivered by a parametrized MPC controller [14], whereby the approximate value function $V_\theta$ is given by

$$\begin{aligned} V_\theta(s) = \min_{X, U, \Sigma} \quad & \lambda_\theta(x_0) + \ell_{f,\theta}(x_N) && \text{(6a)} \\ & \quad + \sum_{k=0}^{N-1} \gamma_{RL}^k \big(\ell_\theta(x_k, u_k) + w^\top \sigma_k\big) \\ \text{s.t.} \quad & x_{k+1} = f_\theta(x_k, u_k), \ k = 0, \dots, N-1, && \text{(6b)} \\ & x_0 = s, && \text{(6c)} \\ & x_k \in \mathcal{S}, \ k = 0, \dots, N, && \text{(6d)} \\ & u_k \in \mathcal{A}, \ k = 0, \dots, N-1, && \text{(6e)} \\ & c_\theta(x_k, u_k) \leq \sigma_k, \quad k = 0, \dots, N-1. && \text{(6f)} \end{aligned}$$

Primal variables include the state $X = \begin{bmatrix} x_0 \dots x_N \end{bmatrix}^\top$ and actions $U = \begin{bmatrix} u_0 \dots u_{N-1} \end{bmatrix}^\top$. The term $\lambda_\theta : \mathcal{S} \to \mathbb{R}$ represents the initial cost, while $\ell_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and $\ell_{f,\theta} : \mathcal{S} \to \mathbb{R}$ denote

the stage and terminal costs. The system dynamics are given by the parameterized model $f_\theta : \mathcal{S} \times \mathcal{A} \to S$. Mixed state–input constraints are given by $c_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, for example the CBF constraint (2), with slack variables $\Sigma = \begin{bmatrix}\sigma_0 \dots \sigma_{N-1}\end{bmatrix}^\top$ ensuring feasibility. Their penalties $w$ control the trade-off between rigidity and relaxation. The resulting policy applies the first optimal control input $\pi_\theta(s) = u_0^\star$. The action-value function $Q_\theta(s,a)$ is defined analogously, as follows:

$$Q_\theta(s,a) = \min_{X,U,\,\Sigma} \quad \text{(6a)} \tag{7a}$$
$$\text{s.t.} \quad \text{(6b)} - \text{(6e)} \tag{7b}$$
$$u_0 = a. \tag{7c}$$

The MPC-based RL framework can be coupled with different RL algorithms; in this work we focus on Q-learning. The objective is to minimize the Bellman residual $\min_\theta \mathbb{E}\left[\|Q^\star(s,a) - Q_\theta(s,a)\|^2\right]$, in order to learn the optimal action-value function and subsequently recover the optimal policy from it. In practice, this amounts to driving the Temporal Difference (TD) error, defined as

$$\tau_k = L(s_k,a_k) + \gamma_{RL} V_\theta(s_{k+1}) - Q_\theta(s_k,a_k), \tag{8}$$

to zero with the parameter update step

$$\theta \leftarrow \theta + \eta \tau_k \nabla_\theta Q_\theta(s_k,a_k), \tag{9}$$

where $\eta$ is the size of the update step [14]–[16].

## III. Methodology

This section introduces the three novel approaches for parametrizing the class $\mathcal{K}$ function of the CBF in MPC-based RL. As a general remark, the methods below parametrize both the objective function and the CBF condition in the MPC. Within these methods, the following two assumptions are considered.

**Assumption 1.** *The prediction model $f$ is known and coincides exactly with the true system dynamics.*

**Assumption 2.** *The CBF is known and its safe set $\mathcal{C} = \{s \in \mathcal{S} \mid h(s) \geq 0\}$ equals the true safe set.*

Lastly, for brevity we define

$$\tilde{J}(X,U,\Sigma) = \ell_{f,\theta}(x_N) + \sum_{k=0}^{N-1} \ell_\theta(x_k,u_k) + w_{\text{MPC}} \sum_{k=0}^{N-1}\sum_{i=1}^{\mathcal{O}} \sigma_{k,i}, \tag{10}$$

where the slack variables $\Sigma = [\sigma_{0,0} \dots \sigma_{N-1,\mathcal{O}}]^T$ for each constraint $i = 1,\dots,\mathcal{O}$ will be later used to ensure feasibility by allowing temporary relaxation of the CBF constraint, penalized in the cost through $w_{\text{MPC}}$. This relaxation is necessary because, under input constraints, the CBF condition (2) may not be feasible for all states $s$, and as such recursive feasibility cannot be guaranteed. Ensuring recursive feasibility is outside the scope of this paper. Instead, feasibility is maintained by allowing temporary relaxation of the CBF constraints, with violations penalized in the RL stage cost. Moreover, during RL exploration, the system may visit states where satisfying the CBF constraint is more likely to be infeasible, highlighting the need for this relaxation mechanism.

### A. Learnable Optimal Decay CBF

A key challenge in the OPTD CBF framework is selecting suitable parameters $\bar{\omega}, P_\omega$. Since this selection is somewhat arbitrary and problem dependent, an RL agent is introduced to tune these quantities for the specific task. Consider a control task with $\mathcal{O}$ constraints (e.g., obstacles). We enforce each constraint $i = 1,\dots,\mathcal{O}$, via a separate CBF, which is in turn associated to learnable parameters $\bar{\omega}_{k,i,\theta}$ and $P_{\omega_{k,i},\theta}$. This leads to the following MPC value function $V_\theta(s)$:

$$\min_{X,\,U,\,\Omega,\,\Sigma} \quad \tilde{J}(X,U,\Sigma) + \sum_{k=0}^{N-1}\sum_{i=1}^{\mathcal{O}} P_{\omega_{k,i},\theta}(\omega_{k,i} - \bar{\omega}_{k,i,\theta})^2 \tag{11a}$$
$$\text{s.t.} \quad \text{(6a)} - \text{(6e)}, \tag{11b}$$
$$h_i(x_{k+1}) - (1-\omega_{k,i})\, h_i(x_k) \geq -\sigma_{k,i}, \quad k = 0,\dots,N-1, i = 1,\dots,\mathcal{O}, \tag{11c}$$
$$\omega_{k,i} \geq 0, \;\; \sigma_{k,i} \geq 0, \quad k = 0,\dots,N-1, i = 1,\dots,\mathcal{O}. \tag{11d}$$

The vector $\Omega = [\omega_{0,0},\dots,\omega_{N-1,\mathcal{O}}]^T$ collects the decision variables that relax the CBF condition at each prediction step and constraint. The decay variables $\omega_{k,i}$ are further shaped by the quadratic penalty function in the objective, which steers them toward the task-adapted reference values $\bar{\omega}_{k,i,\theta}$ while controlling the strength of the penalty for deviations through $P_{\omega_{k,i},\theta}$.

By leveraging an RL algorithm, these parameters can be adjusted to improve closed-loop performance, while the adaptive nature of the decay rate $\omega$ enhances feasibility and promotes minimal safety violations throughout the training process. However, the number of learnable parameters grows with both the MPC prediction horizon and the number of constraints. Every constraint-step pair introduces a decay variable $\omega_{k,i}$ along with its own $\bar{\omega}_{k,i,\theta}$ and $P_{\omega_{k,i},\theta}$. Longer horizons therefore increase not only the total number of CBF constraints and the optimal decay decision variables associated with them, but also the dimensionality of the parameter space to be tuned. Conversely, if the horizon is too short, the RL agent may lack sufficient flexibility to obtain a satisfactory solution. This strong dependence on the horizon is a principal limitation of the method. Lastly, the same formulation can also be applied in settings with time-varying constraints CBFs $h(s_k,k)$.

### B. NN-CBF

To address these limitations, we propose an alternative framework that reduces reliance on the horizon length and provides a richer parametrization of the class $\mathcal{K}$. It removes the optimal-decay decision variables and instead employs a NN to directly learn the decay rates $\gamma$ in the discrete eCBF condition. The inputs to the neural network are the current state $s_k$ and the CBF values, written as $h_i(s_k)$ in the time-invariant case or $h_i(s_k,k)$ when the CBF is time-varying. We also propose to provide as additional input any information that can be beneficial to the decision making, encoded at time step $k$ as the context $c_i(k)$, $i = 1,\dots,\mathcal{O}$. For instance, in an obstacle

avoidance task the context might consist of the position of each obstacle. The network output is instead the vector of decay rates $\Gamma_k^{\text{NN}} = \begin{bmatrix}\gamma_1 & \cdots & \gamma_{\mathcal{O}}\end{bmatrix}^\top$, and is computed as follows:

$$z_k^{(0)} = \begin{bmatrix} x_k & h_1(x_k,k)\cdots h_{\mathcal{O}}(x_k,k) & c_1(k)\cdots c_{\mathcal{O}}(k)\end{bmatrix}^\top, \tag{12}$$

$$z_k^{(i)} = \text{ReLU}\big(W_{i,\theta}\, z_k^{(i-1)} + b_{i,\theta}\big), \quad i = 1,\dots,L, \tag{13}$$

$$\Gamma_k^{\text{NN}} = \text{Sigmoid}\big(W_{L+1,\theta} z_k^{(L)} + b_{L+1,\theta}\big). \tag{14}$$

where $W_{i,\theta}, W_{L+1,\theta}$ and $b_{i,\theta}, b_{L+1,\theta}$ denote the RL-learnable weight matrices and biases, respectively, and $L$ is the number of hidden layers. Applying the network over the entire prediction horizon provides a sequence of decay rates that are substituted into the CBF constraints, constraint-wise and step-wise. Using these outputs in the CBF constraints yields the following parameterized MPC value function $V_\theta(s)$:

$$\min_{X,U,\Sigma} \quad \tilde{J}(X,U,\Sigma) \tag{15a}$$

$$\text{s.t.} \quad \text{(6a)} - \text{(6e)}, \tag{15b}$$

$$h_i\big(x_{k+1}\big) - \big(1-\gamma_{i,k}^{\text{NN}}\big)\, h_i\big(x_k\big) \geq -\sigma_{k,i}, \tag{15c}$$

$$k = 0,\dots,N-1, i = 1,\dots,\mathcal{O},$$

$$\sigma_{k,i} \;\geq\; 0, k = 0,\dots,N-1, i = 1,\dots,\mathcal{O}. \tag{15d}$$

Compared with the LOPTD-CBF, this method is independent of the MPC horizon. This independence is particularly useful for short horizons, where RL can still learn effective policies while the MPC maintains low computational overhead. Moreover, the NN provides a richer parametrization, enabling more expressive nonlinear mappings and finer trade-offs between safety and performance. However, reliance on a NN increases the number of trainable parameters relative to the optimal-decay method and introduces topological and hyperparameter choices that require tuning.

### C. RNN-CBF

Building on the feedforward formulation, we replace the network with an Elman RNN [17] to similarly generate a vector of time-correlated decay rates across the prediction horizon. The RNN is defined by the following equations:

$$z_k^{(0)} = \begin{bmatrix} x_k & h_1(x_k,k)\cdots h_{\mathcal{O}}(x,k) & c_1(k)\cdots c_{\mathcal{O}}(k)\end{bmatrix}^\top, \tag{16}$$

$$q_k^{(1)} = \text{ReLU}\big(W_{i,\theta}\, z_k^{(0)} + b_i + W_{q^{(i)},\theta} q_{k-1}^{(1)}\big), \tag{17}$$

$$q_k^{(i)} = \text{ReLU}\big(W_{i,\theta}\, q_k^{(i-1)} + b_i + W_{q^{(i)},\theta} q_{k-1}^{(i)}\big),\ i = 2,\dots,L, \tag{18}$$

$$\Gamma_k^{\text{RNN}} = \text{Sigmoid}\big(W_{L+1,\theta} z_k^{(L)} + b_{L+1}\big), \tag{19}$$

where $q_k^{(i)}$ denotes the hidden state of layer $i$ at time step $k$. The weight matrices and biases are also RL-learnable, similar to the NN-CBF case. Additionally, the recurrent weight matrices $W_{q^{(i)},\theta}^{(i)}$ also become learnable parameters. The RNN is incorporated into the MPC value function as in (15), by applying the network over the prediction horizon to obtain a sequence of decay rates, as follows:

$$\min_{X,U,\Sigma} \quad \tilde{J}(X,U,\Sigma) \tag{20a}$$

$$\text{s.t.} \quad \text{(6a)} - \text{(6e)}, \tag{20b}$$

$$h_i\big(x_{k+1}\big) - \big(1-\gamma_{i,k}^{\text{RNN}}\big)\, h_i\big(x_k\big) \geq -\sigma_{k,i}, \tag{20c}$$

$$k = 0,\dots,N-1, i = 1,\dots,\mathcal{O},$$

$$\sigma_{k,i} \;\geq\; 0, k = 0,\dots,N-1, i = 1,\dots,\mathcal{O}. \tag{20d}$$

The key idea is that a RNN can learn to store past information in its hidden state. This allows it to remember recent pieces of context information, something a feedforward network does not explicitly capture. As a result, the RNN is able to handle time-varying CBFs more effectively, since its structure is designed to exploit temporal relation. By leveraging their hidden state, RNNs can potentially train more sample-efficiently than feedforward networks [18]. A sufficiently large feedforward network can, in theory, approximate any mapping, but doing so typically requires more training samples than when such temporal structure is inherently imposed, as in an RNN.

Furthermore, since the RNN is rolled out across the MPC horizon by updating its hidden state at each predicted step with the corresponding predicted input. The hidden state is then passed recursively through the architecture. This means each future step has access not only to past trajectory information but also to the sequence of predicted inputs along the horizon. For example, a point at horizon step $k+5$ receives information from the inputs at $k+4$, $k+3$, $k+2$, and so on, through the evolving hidden state. In this way, the CBF conditions at different steps of the horizon are temporally connected through the RNN.

Once the MPC optimization is solved and the first optimal control action $u_0^\star$ is applied, the RNN hidden state is reset to its value before the optimization, $q_k$. It is then updated using the new actual state $s_{k+1}$, so the hidden state remains consistent with the true system trajectory rather than the predicted one.

### D. Training Architecture

The overall training architecture is shown in Algorithm 1, where $N_{\text{episodes}}$ denotes the total number of training episodes. This RL training process is built around three MPCs. The first MPC is based on the value function $V_\theta(s)$ and serves as the behavioral policy for generating online training data. To promote exploration, a perturbation term $\xi^\top u_0$ is added to its objective, where $\xi$ is sampled from a normal distribution. We refer to this modified controller as $V_{\text{rand},\theta}(s,\xi)$. Injecting noise in this manner preserves constraint satisfaction, unlike directly perturbing the applied control input as $a = u_0 + \xi_t$, which may lead to violations. The noise level decays over the course of training, allowing the policy to gradually shift from exploration to exploitation. The second and third MPCs compute $V_\theta(s)$ and $Q_\theta(s,a)$ based on (11) or (15), depending on the choice of parametrization, which are used to form the TD error.

An integral part of the training architecture is the RL stage cost $L(s,a)$, which encodes the return of the task at hand as

per eq. (4). The stage cost here is augmented to account for the use of the slack variables as follows.

$$L_{\text{aug}}(s_k, a_k, \Sigma_k^\star) = L\big(s_k, a_k\big) + w_{\text{RL}} \sum_{k=0}^{N-1} \sum_{i=1}^{\mathcal{O}} \sigma_{k,i}^\star \tag{21}$$

Here, $L(s, a)$ is the RL stage cost and can be chosen to be similar to the MPC stage cost $\ell_\theta(s, a)$ so that RL stage cost remains aligned with the MPC objective. The second term aggregates the optimal slack variables $\Sigma^\star$ from the optimal solution of the MPC $V_{\text{rand},\theta}(s_k, \xi)$. The weight $w_{RL}$ controls how strongly these violations are penalized during learning.

Gradients for the Q-learning update are first accumulated in a buffer and then averaged before updating the parameters. The averaged gradient is computed as

$$\mathbf{g}_{avg} = -\frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \delta_i \nabla_\theta Q_\theta(s_i, a_i), \tag{22}$$

where $|\mathcal{B}|$ denotes the buffer cardinality. In place of the plain gradient descent update (9), we propose to leverage the Adam optimizer instead, which adaptively rescales learning rates by maintaining running estimates of first and second moments of the gradients, often leading to faster and more stable convergence than plain gradient descent [19]. We refer to this modified update as $g_{\text{Adam}}$, with the parameter update given by $\theta \leftarrow \theta - \mathbf{g}_{\text{Adam}}$.

To enforce both lower and upper bounds on the RL parameter updates, the step can be cast as a QP, which projects the updated parameters into the feasible set, as proposed in [13]. In Algorithm 1 the Adam update together with this QP projection is represented by one step called $\text{AdamQP}(\boldsymbol{\theta}, \mathbf{g}_{adam})$. For further details of the code see https://github.com/kerimd14/thesis_code.

**Algorithm 1** RL Training Loop

**for** episode $= 1$ to $N_{\text{episodes}}$ **do**
  Reset the environment and RNN hidden states (if used)
  **for** $t = 0$ to $T - 1$ **do**
    **Exploration:** Solve $V_{\text{rand},\theta}(s_t, \xi_t)$ for $a_t$ and $\Sigma_t^\star$
    **System rollout:** Update $s_{t+1} \leftarrow f(s_t, a_t)$
    Observe stage cost $L_{aug}(s_t, a_t, \Sigma_t^\star)$
    **Q-value:** Solve $Q_\theta(s_t, a_t)$
    **V-value:** Solve $V_\theta(s_{t+1})$
    **TD error:** Compute TD error (8)
    **Compute Gradient:** $\nabla_\theta Q_\theta(s_t, a_t)$,
    Form $\mathbf{g}_t \leftarrow -\tau_t \nabla_\theta Q_\theta(s_t, a_t)$
    Store $\mathbf{g}_t$ in buffer $\mathcal{B}$
  **end for**
  **if** buffer $\mathcal{B}$ is full **then**
    $\mathbf{g}_{avg} \leftarrow \frac{1}{|\mathcal{B}|} \sum \mathbf{g}_t$ and clear buffer $\mathcal{B}$
    Update $\theta \leftarrow \text{AdamQP}(\theta, \mathbf{g}_{avg})$
  **end if**
  Decay exploration noise ($\xi_t$)
**end for**

## IV. Results

We test our proposed methodologies on two obstacle avoidance tasks. The first task evaluates the LOPTD-CBF and the NN-CBF on a simple example with one static obstacle. The second part considers a more complex environment with multiple moving obstacles, where the RNN-CBF and NN-CBF are compared to analyze their relative performance. Both sets of experiments use the following discrete linear 2D double-integrator system as described in [11], with sampling time $\Delta t$ set to 0.2s:

$$s_{k+1} = As_k + Ba_k, \tag{23}$$

In all the results that follows, the control policy is provided by an MPC controller with objective

$$x_N^T P_\theta \, x_N + \sum_{k=0}^{N-1} \big(x_k^T Q \, x_k \; + \; u_k^T R \, u_k), \tag{24}$$

with $Q = 10\mathbf{I}_{4\times4}, R = \mathbf{I}_{2\times2}$, and a learnable terminal weight $P_\theta$ initialized with $100\mathbf{I}_{4\times4}$, constrained to remain diagonal and positive definite. The system is subject to the following state and input constraints: $\mathcal{S} = \big\{s_k \in \mathbb{R}^n : -5\,\mathbf{I}_{4\times1} \leq s_k \leq 5\,\mathbf{I}_{4\times1}\big\}, \mathcal{A} = \big\{a_k \in \mathbb{R}^m : -\mathbf{I}_{2\times1} \leq a_k \leq -\mathbf{I}_{2\times1}\big\}$. A generic control barrier function for a circular obstacle centered at $(x_o, y_o)$ with radius $r_o$ is defined as $h(s) = (s_0 - x_o)^2 + (s_1 - y_o)^2 - r_o^2$, and is used to represent all circular obstacles considered in these experiments. The RL stage cost is chosen with the same reasoning as outlined before in III-D.

### A. Static Obstacle

The first experiment considers the task of avoiding a single circular static obstacle. The system starts at the initial point $(-5, -5)$ and aims to reach the origin while maintaining safety. The single circular obstacle is defined with a center at $(2, 2.25)$ and a radius of $1.5$. To initialize the learning from a suboptimal policy and better appreciate the impact of the RL tuning, the MPC controller is implemented with a horizon of one. This myopic setup is intended to produce a suboptimal baseline policy, which can then be improved upon in subsequent experiments.

*1) Learable Optimal Decay CBF:* We begin by testing the LOPTD-CBF. Prior to training, the initial policy is highly suboptimal, as illustrated in Figure 1a, with a cumulative stage cost of **21712**. Due to the one-horizon MPC-CBF, the policy is highly myopic and fails to anticipate the obstacle. Consequently, the system moves diagonally toward the goal until it reaches the obstacle boundary, after which it follows the boundary to maintain safety and satisfy the CBF condition.

After training, the cumulative stage cost decreases to **7156**. As shown in Figure 1b, the trajectory no longer moves directly toward the goal but instead deviates left early to avoid the obstacle preemptively. The system also reaches the target more efficiently, achieving a higher velocity and requiring fewer iterations to converge.

The improved behavior is also evident in the $\omega$ dynamics (Figure 1e). A pronounced spike appears near the obstacle, reflecting the greater relaxation needed to maintain feasibility

at higher velocity. In other words, as the MPC controller drives the system more quickly, satisfying the CBF constraint necessitates a larger decay rate $\omega$.

These changes are directly influenced by the learned terminal cost parameters, shown in Figure 1h. The position weights ($P_{1,1}$ and $P_{2,2}$) dominate the velocity weights ($P_{3,3}$ and $P_{4,4}$), encouraging the system to prioritize reaching the origin quickly. Moreover, since $P_{3,3}$ is larger than $P_{4,4}$, motion in the $x$-direction is penalized more heavily. As a result, movement along the $y$-axis is comparatively cheaper, leading the system to steer left around the obstacle.

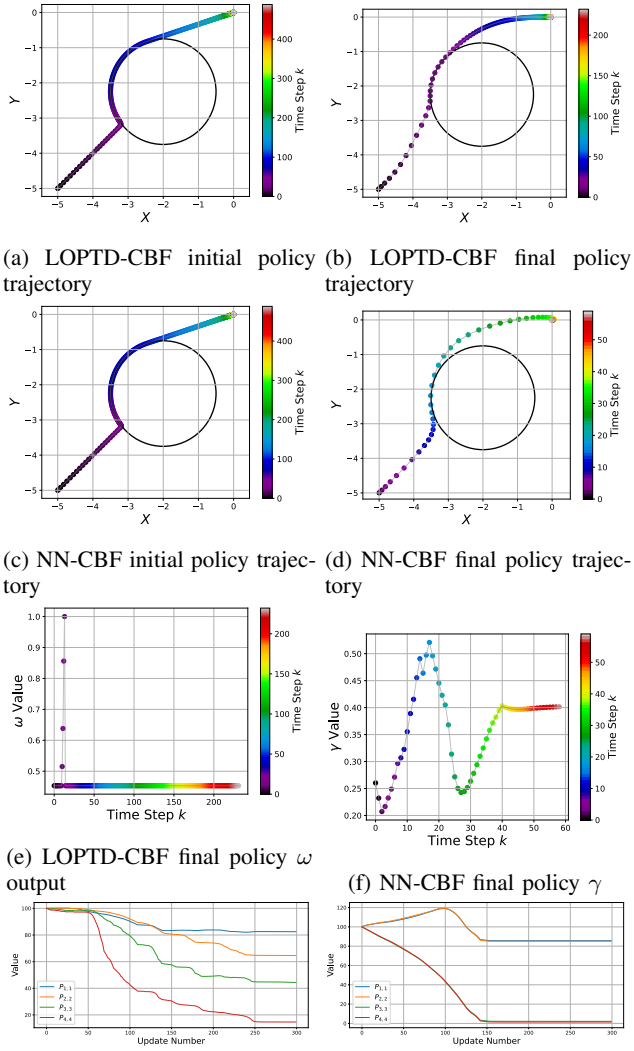
*2) Neural Network CBF:* For the NN-CBF, the initial policy is again suboptimal, with a cumulative stage cost of **21892** (Figure 1c), as it remains too myopic to anticipate the obstacle. After training, the cumulative stage cost decreases to **6627**, with the improved policy shown in Figure 1d.

The improved behavior can be explained by Figure 1f, which shows the evolution of $\gamma$, the NN output. At the start, $\gamma$ decreases, making the policy more conservative and preventing rapid acceleration. It then increases, enabling faster progress toward the goal. Near the obstacle, $\gamma$ decreases again, introducing conservatism that facilitates a safe turn around the circle. After passing the obstacle, $\gamma$ rises to permit faster motion, before decreasing once more as the system approaches the target, ensuring smooth deceleration and preventing overshoot. Finally, $\gamma$ stabilizes, allowing the system to settle at the goal.

The terminal cost parameters also influence the outcome. As shown in Figure 1h, the position weights dominate the velocity weights, promoting faster motion, while the nearly equal velocity weights encourage forward rather than leftward steering. Nevertheless, the trajectory still turns left, indicating that the deviation is shaped not only by the terminal cost but also by the CBF constraint.

*3) Comparison:* Both the LOPTD-CBF and the NN-CBF improve over the initial MPC-CBF policy, but each has distinct advantages and limitations. The LOPTD-CBF requires fewer learnable parameters, which makes training faster since the RL agent optimizes over a smaller search space. Another advantage of the LOPTD-CBF is that it introduces a decision variable for safety directly into the MPC optimization problem. This allows online adjustment of the trajectory and safety, reducing the risk of safety violations during training. On the other hand, its performance is strongly dependent on the MPC horizon, as a shorter horizon leads to a smaller number of $\omega$ variables, thus restricting the adaptability of the CBF condition only to imminent states, rendering it more myopic.

The NN-CBF, in contrast, employs more trainable parameters, enabling the representation of more complex class $\mathcal{K}$ functions and the generation of state-dependent decay rates. Owing to its richer parameterization, the NN-CBF is less dependent on the horizon length for tuning the decay rate compared to the LOPTD-CBF. The combination of reduced dependence on the horizon and richer safety representation leads to lower cumulative stage cost compared to the LOPTD-CBF. However, while the NN-CBF is evaluated at every timestep within a rollout, its outputs (decay rates) are determined by the fixed, pre-trained network parameters and cannot adapt online during an episode. This limitation can compromise safety when the network is poorly initialized, unlike the LOPTD-CBF which can adjust its decay rates adaptively.




(a) LOPTD-CBF initial policy trajectory

(b) LOPTD-CBF final policy trajectory

(c) NN-CBF initial policy trajectory

(d) NN-CBF final policy trajectory

(e) LOPTD-CBF final policy $\omega$ output

(f) NN-CBF final policy $\gamma$

(g) Evolution of terminal cost matrix $P$ during training LOPTD-CBF

(h) Evolution of terminal cost matrix $P$ during training NN-CBF

Fig. 1: Static Obstacle Experiment Results

## B. Dynamic Obstacles

To evaluate the framework in the presence of time-varying obstacles, an experiment was conducted involving two dynamic obstacles and one static obstacle. The system starts at $(-5, -5)$ and must reach the origin while safely navigating between the moving obstacles. The two dynamic obstacles move horizontally along the $x$-axis within predefined bounds. The first obstacle moves at $y_o = -1.5$ within $x_o \in [-4.0, 0.0]$, and the second moves at $y_o = -3.3$ within $x_o \in [-4.0, 1.0]$. Both move back and forth at constant velocity and have a

radius of 0.7. A third static circular obstacle is placed at $(x_o, y_o) = (-2, 0)$ with a radius of 1. The two dynamic obstacles hinder the system as it moves toward the origin, while the static obstacle discourages it from going around and forces the controller to find a trajectory that passes between the moving obstacles.

*1) Neural Network CBF:* Figure 2 shows snapshots of the NN-CBF policy before and after training. Initially, the trajectory is unsafe, entering the first dynamic obstacle at $k = 16$ and $k = 17$, and the MPC predictions also fail to avoid future obstacle positions. After training, the trajectory slows earlier, remains outside the obstacle at $k = 16$ to $k = 18$, and maneuvers safely around it. The predicted horizon also avoids the future positions of the dynamic obstacle, indicating improved planning.

This improvement is also reflected in the NN outputs shown in Figure 4a. The largest change occurs in $\gamma_1$ for the first dynamic obstacle (blue), whose constraint was previously violated. After training, $\gamma_1$ decreases, corresponding to a more conservative decay rate. This adjustment causes the system to slow earlier when approaching the obstacle. The sharpest drop appears between $k = 8$ and $k = 15$, aligning with the trajectory change in Figure 2. For example in Figure 2, at $k = 11$ the predicted state trajectory now avoids the predicted obstacles by slowing down much earlier.

Achieving this behavior still requires slack variables. Slacks are introduced during training to temporarily allow constraint violations, while penalties on their use encourage the MPC to reduce violations and approach a safe, control-invariant solution. In Figure 2, slacks are not used to bypass constraints but to enable rapid deceleration under bounded inputs. This becomes important when obstacles move faster or input bounds are tight. The trajectory yields a cumulative stage cost of **6067** when excluding the slack penalty term and **15194** when including it, as defined by the augmented stage cost in (21).

*2) Recurrent Neural Network CBF:* Figure 3 shows the RNN-CBF policy before and after training. Initially the trajectory is once again unsafe, entering the obstacle at $k = 16$ to $k = 17$. After training, the policy becomes safe, similar to the NN-CBF case, though less conservative, as the trajectory passes closer to the obstacles at $k = 17$.

The RNN outputs follow the same trend as in the NN case. The $\gamma_1$ values for the first dynamic obstacle (blue) decrease between $k = 8$ and $k = 15$, but less sharply than in the NN, leaving a higher decay rate and producing less conservative behavior, as seen at $k = 16$ in Figure 3. As with the NN-CBF, a richer RNN parametrization could in principle achieve this without slacks. The trajectory has a cumulative stage cost of **5923** without slack penalties and **14026** when they are included, as computed using the augmented stage cost in (21).

*3) Comparison:* The main difference between the NN-CBF and RNN-CBF lies in training efficiency. The RNN learns the same policy with roughly 170 fewer episodes, highlighting the benefit of its recurrent structure. While further tests are needed to confirm, these results suggest that the RNN learns faster than the NN. The RNN trajectory is also more effective, coming closer to obstacles and achieving a lower stage cost.

A limitation of both methods is that gradient updates occur only when the CBF constraint is active. When inactive, the associated dual variable in the Lagrangian of $Q_\theta(s, a)$ is zero, so no gradient flows to the network. Combined with the nonlinearities introduced by the NNs, this increases the difficulty for the interior-point solver. Although, such solvers can handle nonlinear problems, larger networks and longer horizons make the optimization more challenging.

## V. Conclusion

In this paper we proposed novel methods for merging MPC, RL and CBFs. The central idea is to use a parameterized MPC scheme, with CBF constraints, as a function approximator to learn a safer and more performing policy. The proposed methods expanded on this idea by exploring different ways of parameterizing and adapting the class $\mathcal{K}$ function in the CBF condition. The first method, LOPTD-CBF, focused on improving feasibility guarantees while allowing RL to search for better policies. This was achieved by extending and parameterizing the OPTD framework. The second method, NN-CBF, introduced a richer parametrization through a NN, enabling more expressive safety conditions and yielding policies with better performance than LOPTD-CBF. Finally, the RNN-CBF extended the NN-CBF framework with a recurrent structure to better handle moving obstacles by accounting for temporal dependencies. Numerical experiments confirmed that both LOPTD-CBF and NN-CBF produce safe and improved policies in a simple obstacle avoidance scenario, with NN-CBF outperforming LOPTD-CBF. In the case of time-varying obstacles, additional experiments showed that RNN-CBF is able to find a safe policy faster than NN-CBF. However, both solutions required the use of slack variables, even though a slack-free policy is theoretically attainable but was not demonstrated in this work. Further research will focus on addressing the limitations of the current work, particularly regarding gradient updates and nonlinearities in the MPC network.

## References


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

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

[3] A. E. Chriat and C. Sun, "On the optimality, stability, and feasibility of control barrier functions: An adaptive learning-based approach," *IEEE Robotics and Automation Letters*, vol. 8, no. 11, pp. 7865–7872, 2023.

[4] J. Zeng, Z. Li, and K. Sreenath, "Enhancing feasibility and safety of nonlinear model predictive control with discrete-time control barrier functions," in *2021 60th IEEE Conference on Decision and Control (CDC)*, 2021, pp. 6137–6144.

[5] W. Xiao, T.-H. Wang, R. Hasani, M. Chahine, A. Amini, X. Li, and D. Rus, "Barriernet: Differentiable control barrier functions for learning of safe robot control," *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 2289–2307, 2023.

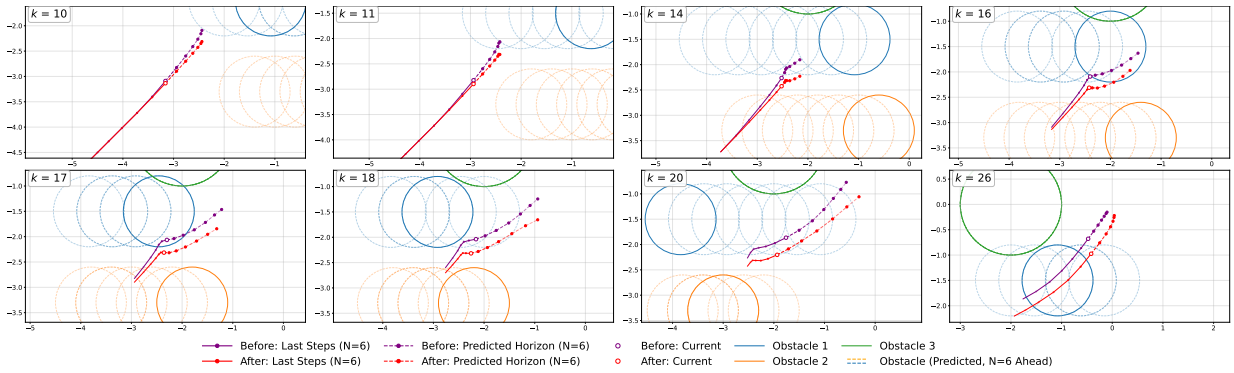



Fig. 2: Snapshots of the NN-CBF policy, where 'before' refers to before training and 'after' refers to after training.

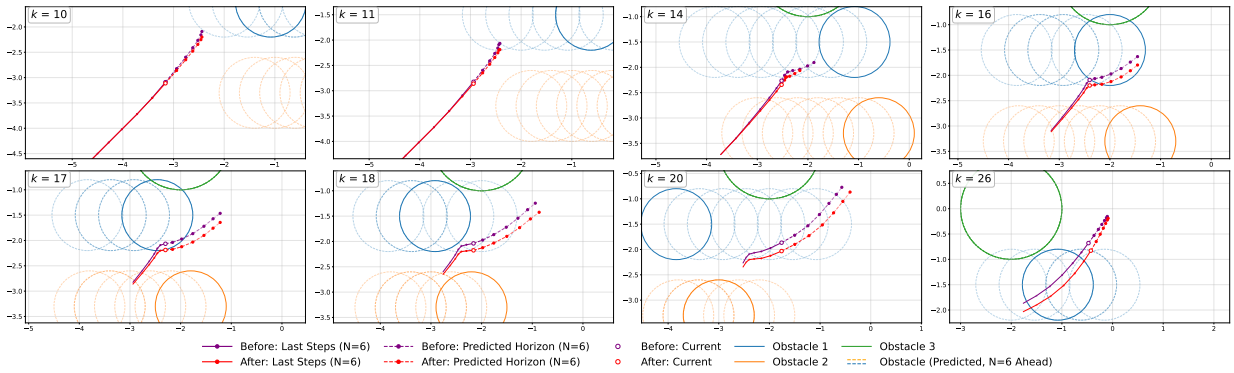
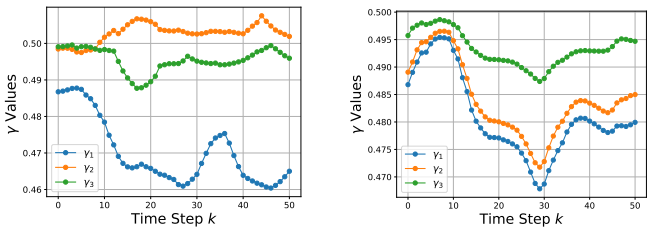



Fig. 3: Snapshots of the RNN-CBF policy, where 'before' refers to before training and 'after' refers to after training.




(a) $\gamma_i^{NN}$ values of the NN (b) $\gamma_i^{RNN}$ values of the RNN

Fig. 4: NN and RNN outputs after training


[6] E. Sabouni, H. Sabbir Ahmad, V. Giammarino, C. G. Cassandras, I. C. Paschalidis, and W. Li, "Reinforcement learning-based receding horizon control using adaptive control barrier functions for safety-critical systems*," in *2024 IEEE 63rd Conference on Decision and Control (CDC)*, 2024, pp. 401–406.

[7] F. Airaldi, B. De Schutter, and A. Dabiri, "Probabilistically safe and efficient model-based reinforcement learning," *arXiv preprint arXiv:2504.00626*, 2025.

[8] T. Kim, R. I. Kee, and D. Panagou, "Learning to refine input constrained control barrier functions via uncertainty-aware online parameter adaptation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2025, pp. 3868–3875.

[9] J. Zeng, B. Zhang, Z. Li, and K. Sreenath, "Safety-critical control using optimal-decay control barrier function with guaranteed point-wise feasibility," in *2021 American Control Conference (ACC)*, 2021, pp. 3856–3863.

[10] A. Agrawal and K. Sreenath, "Discrete control barrier functions for safety-critical control of discrete systems with application to bipedal robot navigation," in *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.

[11] J. Zeng, B. Zhang, and K. Sreenath, "Safety-critical model predictive control with discrete-time control barrier function," in *2021 American Control Conference (ACC)*, 2021, pp. 3882–3889.

[12] Q. Nguyen and K. Sreenath, "Exponential control barrier functions for enforcing high relative-degree safety-critical constraints," in *2016 American Control Conference (ACC)*, 2016, pp. 322–328.

[13] F. Airaldi, B. D. Schutter, and A. Dabiri, "Reinforcement learning with model predictive control for highway ramp metering," *IEEE Transactions on Intelligent Transportation Systems*, vol. 26, no. 5, pp. 5988–6004, 2025.

[14] S. Gros and M. Zanon, "Data-driven economic NMPC using reinforcement learning," *IEEE Transactions on Automatic Control*, vol. 65, no. 2, pp. 636–648, 2020.

[15] C. J. Watkins, "Learning from delayed rewards," PhD thesis, King's College, Cambridge, 1989.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[17] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[18] F. Bonassi, M. Farina, J. Xie, and R. Scattolini, "On recurrent neural networks for learning-based control: Recent results and ideas for future developments," *Journal of Process Control*, vol. 114, pp. 92–104, 2022.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations (ICLR)*, 2015.

# Bibliography


[1] Ayush Agrawal and Koushil Sreenath. Discrete control barrier functions for safety-critical control of discrete systems with application to bipedal robot navigation. In *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.

[2] Devansh R. Agrawal and Dimitra Panagou. Safe control synthesis via input constrained control barrier functions. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 6113–6118, 2021.

[3] Filippo Airaldi, Bart De Schutter, and Azita Dabiri. Probabilistically safe and efficient model-based reinforcement learning. *arXiv preprint arXiv:2504.00626*, 2025.

[4] Filippo Airaldi, Bart De Schutter, and Azita Dabiri. Learning safety in model-based reinforcement learning using MPC and Gaussian Processes. *IFAC*, 56(2):5759–5764, 2023. 22nd IFAC World Congress.

[5] Filippo Airaldi, Bart De Schutter, and Azita Dabiri. Reinforcement learning with model predictive control for highway ramp metering. *IEEE Transactions on Intelligent Transportation Systems*, 26(5):5988–6004, 2025.

[6] Anil Alan, Tamas G. Molnar, Aaron D. Ames, and Gábor Orosz. Parameterized barrier functions to guarantee safety under uncertainty. *IEEE Control Systems Letters*, 7:2077–2082, 2023.

[7] Anil Alan, Andrew J. Taylor, Chaozhe R. He, Aaron D. Ames, and Gábor Orosz. Control barrier functions and input-to-state safety with application to automated vehicles. *IEEE Transactions on Control Systems Technology*, 31(6):2744–2759, 2023.

[8] Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *Proceedings of the 18th European Control Conference (ECC)*, pages 3420–3431, 2019.

[9] Aaron D. Ames, Gennaro Notomista, Yorai Wardi, and Magnus Egerstedt. Integral control barrier functions for dynamically defined control laws. *IEEE Control Systems Letters*, 5(3):887–892, 2021.

[10] 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, 2017.

[11] Fabio Bonassi, Marcello Farina, Jing Xie, and Riccardo Scattolini. On recurrent neural networks for learning-based control: Recent results and ideas for future developments. *Journal of Process Control*, 114:92–104, 2022.

[12] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst. *Reinforcement Learning and Dynamic Programming using Function Approximators*. CRC Press, 2017.

[13] Alaa Eddine Chriat and Chuangchuang Sun. On the optimality, stability, and feasibility of control barrier functions: An adaptive learning-based approach. *IEEE Robotics and Automation Letters*, 8(11):7865–7872, 2023.

[14] Andrew Clark. Control barrier functions for stochastic systems. *Automatica*, 130:109688, 2021.

[15] Ersin Das and Joel W Burdick. Robust control barrier functions using uncertainty estimation with application to mobile robots. *arXiv preprint arXiv:2401.01881*, 2024.

[16] 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. *IEEE Transactions on Robotics*, 39(3):1749–1767, 2023.

[17] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[18] Sébastien Gros and Mario Zanon. Data-driven economic NMPC using reinforcement learning. *IEEE Transactions on Automatic Control*, 65(2):636–648, 2020.

[19] Hassan K. Khalil. *Nonlinear Control*. Pearson Education Limited, Harlow, England, global edition, 2014.

[20] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

[21] Shishir Kolathaya and Aaron D. Ames. Input-to-state safety with control barrier functions. *IEEE Control Systems Letters*, 3(1):108–113, 2019.

[22] M. Nagumo. über die lage der integralkurven gewöhnlicher differentialgleichungen. *Proceedings of the Physico-Mathematical Society of Japan. 3rd Series*, 24:551–559, 1942.

[23] Allan Andre do Nascimento, Antonis Papachristodoulou, and Kostas Margellos. Probabilistically safe controllers based on control barrier functions and scenario model predictive control. *arXiv preprint arXiv:2409.06834*, 2024.

[24] Quan Nguyen and Koushil Sreenath. Exponential control barrier functions for enforcing high relative-degree safety-critical constraints. In *2016 American Control Conference (ACC)*, pages 322–328, 2016.

[25] J. Rawlings and D. Mayne. *Model Predictive Control: Theory and Design*. Nob Hill Publishing, 2008.

[26] Ehsan Sabouni, H.M. Sabbir Ahmad, Vittorio Giammarino, Christos G. Cassandras, Ioannis Ch. Paschalidis, and Wenchao Li. Reinforcement learning-based receding horizon control using adaptive control barrier functions for safety-critical systems*. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pages 401–406, 2024.

[27] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML)*, volume 32, pages 387–395. PMLR, 2014.

[28] Oswin So, Zachary Serlin, Makai Mann, Jake Gonzales, Kwesi Rutledge, Nicholas Roy, and Chuchu Fan. How to train your neural control barrier function: Learning safety filters for complex input-constrained systems. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11532–11539, 2024.

[29] Eduardo D. Sontag. A 'universal' construction of Artstein's theorem on nonlinear stabilization. *Systems  Control Letters*, 13(2):117–123, 1989.

[30] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[31] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. Published online 28 April 2005.

[32] Christopher J.C.H. Watkins. *Learning from Delayed Rewards*. Phd thesis, King's College, Cambridge, 1989.

[33] Wei Xiao and Calin Belta. High-order control barrier functions. *IEEE Transactions on Automatic Control*, 67(7):3655–3662, 2022.

[34] 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, 2023.

[35] Yuhan Xiong, Di-Hua Zhai, Mahdi Tavakoli, and Yuanqing Xia. Discrete-time control barrier function: High-order case and adaptive case. *IEEE Transactions on Cybernetics*, 53(5):3231–3239, 2023.

[36] Jun Zeng, Zhongyu Li, and Koushil Sreenath. Enhancing feasibility and safety of nonlinear model predictive control with discrete-time control barrier functions. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 6137–6144, 2021.

[37] Jun Zeng, Bike Zhang, Zhongyu Li, and Koushil Sreenath. Safety-critical control using optimal-decay control barrier function with guaranteed point-wise feasibility. In *2021 American Control Conference (ACC)*, pages 3856–3863, 2021.

[38] Jun Zeng, Bike Zhang, and Koushil Sreenath. Safety-critical model predictive control with discrete-time control barrier function. In *2021 American Control Conference (ACC)*, pages 3882–3889, 2021.

# Glossary

## List of Acronyms

**MPC** Model Predictive Control

**NMPC** Nonlinear Model Predictive Control

**RL** Reinforcement Learning

**CBF** Control Barrier Function

**ZCBF** Zeroing Control Barrier Function

**CLF** Control Lyapunov Function

**eCBF** Exponential Control Barrier Function

**OPTD-CBF** Optimal-Decay Control Barrier Function

**HOCBF** Higher-Order Control Barrier Function

**MPC-CBF** Model Predictive Control with Control Barrier Functions

**CLF-CB-FQP** Control Lyapunov Function–Control Barrier Function Quadratic Program

**QP** Quadratic Program

**NN** Neural Network

**RNN** Recurrent Neural Network

**NN-CBF** Neural Network Control Barrier Function

**RNN-CBF** Recurrent Neural Network Control Barrier Function

**LOPTD-CBF** Learnable Optimal Decay Control Barrier Function

**ReLU** Rectified Linear Unit

**TD** Temporal-Difference

**KKT** Karush–Kuhn–Tucker

**Adam** Adaptive Moment Estimation

**IPOPT** Interior Point OPTimizer

## List of Symbols

$\alpha(\cdot)$ Class $\mathcal{K}$ function
$\bar{\omega}$ Reference decay rate (OPTD-CBF)
$\beta_1, \beta_2, \varepsilon$ Adam optimizer hyperparameters
$\chi$, $\mu$, $\nu$, $\zeta$ Lagrange multipliers
$\Delta t$ Sampling time
$\ell(x, u)$ MPC stage cost
$\ell_\theta(x, u)$, $\ell_{f,\theta}(x)$ Parametrized stage/terminal costs
$\ell_f(x_N)$ Terminal cost
$\eta$ Learning rate
$\gamma_k$ Discrete eCBF decay rate at step $k$
$\gamma_{RL}$ RL discount factor
$\hat{m}_t$, $\hat{v}_t$ Bias-corrected Adam moments
$\lambda_\theta(x_0)$ Initial cost term in parametric MPC
$\mathbf{y}$ Primal–dual variable vector $(x, u, \chi, \mu, \nu, \zeta)$
$\mathcal{A}$ Action space
$\mathcal{L}_\theta(s, a, \mathbf{y})$ Lagrangian of the action–value MPC problem
$\mathcal{O}$ Number of obstacles
$\mathcal{S}$ State space
$\mathcal{U}$ Input constraint set
$\mathcal{X}$ State constraint set
$\mathcal{X}_f$ Terminal constraint set
$\mathrm{NN}_\theta(\cdot)$ Parametrized neural network
$\mathrm{ReLU}(\cdot)$ Rectified Linear Unit activation
$\mathrm{RNN}_\theta(\cdot)$ Parametrized recurrent neural network for decay-rate mapping
$\nabla_\theta(\cdot)$ Gradient w.r.t. $\theta$
$\Omega$ Vector of decay variables over horizon
$\omega$ Optimal-decay variable (OPTD-CBF)
$\partial C$ Boundary of the safe set
$\pi(s)$ Policy mapping state to action
$\pi^\star$ Optimal policy
$\psi(\cdot)$ Penalty function on decay-variable deviation

$\psi_i(x)$ HOCBF auxiliary functions
$\rho(W)$ Spectral radius
$\sigma$ CLF slack variable
$\sigma_k$ CBF slack at step $k$
$\sigma_k$, $\sigma_N$ Slack variables for inequality/terminal constraints
$\sigma_{k,i}$ CBF slack for obstacle $i$ at step $k$
$\tau_k$ Temporal-difference (TD) error
Sigmoid$(\cdot)$ Sigmoid function
$\theta$ Parameter vector
$\varphi(\cdot)$ Penalty on CLF slack
$a$ Action
$a'$ Next action
$C = \{x \in \mathbb{R}^n \mid h(x) \geq 0\}$ Safe set
$c_\theta(x, u)$, $c_\theta^f(x)$ Parametrized mixed/terminal constraints
$c_i(k)$ Context of CBF $i$ at time $k$
$f(x)$ Autonomous state dynamics (continuous time)
$f_\theta(x, u)$ Parametrized dynamics model
$g(x)$ Control input dynamics (continuous time)
$h(\cdot)$ Barrier function/Control barrier function
$H(x)$ QP weight matrix on control effort
$h_i(x_k, k)$ Time-varying barrier function/ control barrier function
$J(x_k)$ MPC objective at time $k$
$k(x)$ Nominal controller in OPTD-CBF objective
$k_b$ eCBF gain in $\alpha(h) = k_b h$
$L(s, a)$ RL stage cost
$L_f h(x)$, $L_g h(x)$ Lie derivatives of $h$ along $f$ and $g$
$L_f V(x)$, $L_g V(x)$ Lie derivatives of $V$ along $f$ and $g$
$L_f^r h(x)$ Higher-order Lie derivative of $h$
$m_t$, $v_t$ Adam first and second moment estimates
$M_{\text{CBF}}$, $M_{\text{CLF}}$ Horizon lengths for CBF/CLF constraints
$O(\cdot)$ Higher-order terms used in BarrierNet constraints
$P$ Terminal cost matrix
$p$ Penalty weight on $\delta^2$ in QP objective
$p(z)$ BarrierNet penalty function
$P[s' \mid s, a]$ State–transition kernel
$P_\omega$ Penalty on $(\omega - \bar{\omega})^2$
$Q(s, a)$ Action–state value function
$Q^\star(s, a)$ Optimal action–state value function
$Q_\theta(s, a)$, $V_\theta(s)$ Parametric value functions
$q_k$ RNN hidden state

| | |
|---|---|
| $r$ | Relative degree |
| $s$ | Current state (RL) |
| $s'$ | Next state |
| $u_0^\star$ | First optimal control input |
| $u_k$ | Control input of MPC |
| $u_{\min}, u_{\max}$ | Input bounds |
| $u_{RL}$, $u_r$ | RL action and projected safe action |
| $V(s)$ | State–value function |
| $V(x)$ | Lyapunov / CLF candidate |
| $V^\star(s)$ | Optimal state–value function |
| $w$, $w_f$ | Slack weights for constraint slacks in MPC objective |
| $W_i$, $b_i$ | Layer weights and biases |
| $w_{\text{MPC}}$ | Slack penalty weight in MPC objective |
| $x_k$ | State vector of MPC |
| $x_N$ | Terminal state vector |
| $z^{(i)}$ | Layer activations |
| $z_{k,i}$ | Auxiliary decision variable introduced in the CBF constraint at prediction step $k$ for constraint $i$, used to tighten the barrier condition. |