Safe Reinforcement Learning Using Sequential Constraints for Collision Avoidance

Gyuyong Hwang [1] and Gyuho Eoh [2,*]

[1] Department of Electronics Engineering, Tech University of Korea, Siheung-si, Gyeonggi-do 15073, South Korea
[2] Department of Mechatronics Engineering, Tech University of Korea, Siheung-si, Gyeonggi-do 15073, South Korea
Email: 1213hgy@tukorea.ac.kr (G.H.); gyuho.eoh@tukorea.ac.kr (G.E.)
*Corresponding author

***Abstract*—Ensuring a high level of safety is essential for collision avoidance in real-world robotic applications. Traditional Reinforcement Learning (RL)-based collision avoidance methods offer adaptability but lack safety guarantees, especially in uncertain and dynamic environments. To address this, we propose a novel safe reinforcement learning (SafeRL) framework called Control Recovery and Barrier Function (CRBF), which enhances safety by sequentially applying different control strategies based on the robot's proximity to obstacles. The CRBF categorizes risk into three distinct levels and adaptively switches between a vanilla RL-based policy, Control Barrier Function (CBF), and a Recovery Function (RF) to prevent collisions and recover from critical situations. In addition, we introduce a constraint-aware training strategy that incorporates these sequential safety mechanisms during policy updates. We validate our method in both simulated and real-world environments, where CRBF outperforms conventional methods, with improvements of up to 22.5% in collision avoidance success rates, particularly in challenging dynamic scenarios.**

***Keywords*—collision avoidance, safe reinforcement learning, control barrier functions, recovery function**

I. Introduction

Robots are increasingly deployed in real-world environments, where collision avoidance is essential to ensure both operational success and safety [1, 2]. Over the past few decades, numerous collision avoidance approaches have been proposed, which are broadly categorized into trajectory-based and reactive-based methods. Trajectory methods explore an optimal path in advance based on global information but struggle with unforeseen obstacles due to their reliance on pre-planned paths [3]. In contrast, reactive methods respond to local changes in real time, providing better adaptability in dynamic environments [4]. For example, reactive techniques, such as Artificial Potential Fields (APFs) [5] and collision cone methods [6], rely on rule-based or short-term responses. While reactive methods offer fast reaction times, they lack long-term adaptability and often fail in complex scenarios.

To address these limitations, Reinforcement Learning (RL) has been increasingly applied to robotic navigation [7]. RL allows agents to learn optimal behaviors through trial-and-error interaction with the environment, making it well suited to dynamic and uncertain environments. However, RL methods face significant safety challenges during training and deployment, due to model uncertainty and the stochastic nature of real-world environments [8–10]. Safe Reinforcement Learning (SafeRL) has emerged to address this issue by incorporating safety constraints into the learning process [11]. Among these, Control Barrier Function (CBF) are widely used to formally constrain the system from entering unsafe states [12, 13]. However, the CBF typically assume known dynamics and cannot ensure recovery if the system has already violated the safety margin [14, 15].

To overcome the above critical limitation, we propose a novel SafeRL framework called Control Recovery and Barrier Function (CRBF). The CRBF introduces a sequential control strategy based on the robot's proximity to obstacles and classifies operational domains into three risk areas: *safe*, *risky*, and *critical*. In the safe area, where there are no nearby obstacles, the robot directly follows the vanilla RL-based policy. In the risky area, where nearby obstacles are detected, the robot's actions are filtered by a CBF to ensure safety. In the critical area, where the risk of collision is imminent, a Recovery Function (RF) overrides the control to guide the robot away from the danger and back to a safe state. This hierarchical design allows our CRBF-based SafeRL to adaptively switch control strategies, ensuring safety even in uncertain and dynamic environments. Unlike previous works that use a single safety mechanism (e.g., CBF or recovery alone), our method dynamically switches between RL, CBF, and RF based on real-time risk assessment, enabling both prevention and recovery in a unified control architecture.

The main contributions of this paper can be summarized as follows:

- We propose a novel SafeRL framework called CRBF, which sequentially combines CBF and a RF based on the robot's proximity to obstacles.

Manuscript received February 11, 2025; revised March 25, 2025; accepted April 16, 2025; published June 17, 2025.

- We define three risk domains to dynamically switch between RL, CBF, and RF policies.
- We introduce a CRBF-based training strategy that integrates sequential constraints into policy updates by combining raw actions, penalty corrections, and recovery-guided adjustments.
- We demonstrate the effectiveness of our method in both simulated and real-world environments.

The remainder of the paper is organized as follows. Section II reviews related work relevant to our study. Section III provides preliminary background information. In Section IV, we formulate the problem addressed in this paper. Section V introduces the definition of danger levels, which represent the robot's proximity-based risk levels. Section VI details the proposed CRBF framework and its control components. In Section VII, we describe the training setup and propose a CRBF-based SafeRL training strategy that incorporates mixed policy updates. Section VIII presents the results of simulation experiments comparing our method with existing collision avoidance algorithms. Section IX provides the results of real-world experiments conducted to validate our approach. Section X offers a discussion of our findings and outlines future research directions. Finally, Section XI concludes the paper and summarizes key contributions and potential avenues for further investigation.

II. Related Works

A. Reinforcement Learning-Based Collision Avoidance

RL-based collision avoidance allows robots to learn navigation strategies by interacting with the environment. It has been shown to be effective in several real-world applications, including autonomous vehicles [16], unmanned aerial vehicles [17], and maritime systems [18]. For example, Long *et al.* [19] proposed a decentralized multi-robot control system based on deep reinforcement learning equipped with onboard sensors and demonstrated scalability to over a hundred robots in densely populated environments. Liang *et al.* [20] focused on collision avoidance in dense and confined spaces and successfully transferred simulation-trained policies to previously unseen real-world scenarios. Everett *et al.* [21] developed an RL-based collision avoidance algorithm that does not rely on predefined behavioral rules, and demonstrated strong generalization

Meanwhile, to improve robustness under uncertainty, Roghair *et al.* [22] improved collision avoidance performance by integrating multiple algorithms to process noisy visual inputs. Kahn *et al.* [8] introduced uncertainty-aware RL by incorporating confidence estimates into the decision-making process, enabling safer navigation in stochastic and partially observable environments.

B. Safe Reinforcement Learning

SafeRL addresses the safety limitations of conventional RL by incorporating constraints or risk-sensitive mechanisms during policy learning and execution. Lütjens *et al.* [23] improved safety by estimating model uncertainty, while Srouji *et al.* [24] combined emergency braking with RL to eliminate collisions during training. Zhou *et al.* [25] improved robustness by decoupling safety control from goal achievement.

Recent advances have integrated CBF into RL frameworks to enforce formal safety guarantees. CBF ensures safety by enforcing state constraints and have shown strong performance in robotic systems [14], especially when combined with RL in complex environments [26, 27]. Cheng *et al.* [28] showed that CBF-based SafeRL can maintain high safety probabilities in continuous control tasks, and Cai *et al.* [29] extended the approach to multi-agent systems. To further improve robustness under uncertainty, Emam *et al.* [30] introduced a robust CBF layer that accounts for worst-case perturbations, and Hu *et al.* [16] applied Gaussian processes to model uncertainty and enforce probabilistic safety guarantees.

However, most of these approaches either lack recovery mechanisms after a safety violation or are limited to simulation environments. The proposed method introduces a CRBF that adaptively combines RL, CBF, and a recovery policy to enable both preventive and corrective safety behaviors in real-world environments.

III. Preliminaries

A. Reinforcement Learning

RL can be formulated by Markov Decision Processes (MDP) $\mathcal{M}$ as follows:

$$\mathcal{M} = \langle S, A, P, R, \gamma \rangle, \tag{1}$$

where S and A are the state and action spaces, respectively. The state transition probability $P(s'|s,a) \to \mathbb{R}$ is described by the state $s \in \mathrm{S}$ and the action $a \in \mathrm{A}$. The $R(s,a)$ is the reward function and $\gamma \in [0,1]$ is the discount factor. The goal of RL is to compute the optimal policy π^* by maximizing the expected cumulative rewards under the policy π as follows:

$$\pi^* = arg\max_{\pi \in \Pi} \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right], \tag{2}$$

where $s_t \sim P(\cdot\,|s_{t-1}, a_{t-1})$ is the state transition, $a_t \sim \pi(\cdot\,|s_t)$ is selected according to the policy π. Bellman equation for state-action value function $\mathrm{Q}^\pi(s,a)$ is used to evaluate the policy:

$$\mathrm{Q}^\pi(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim \mathrm{P}(\cdot|s,a)} \left[\mathbb{E}_{a' \sim \pi(\cdot|s')} [\mathrm{Q}^\pi(s',a')] \right]. \tag{3}$$

Finally, the goal of RL is to compute the optimal policy π^* from the optimal state-action value function $Q^*(s,a)$ as follows:

$$\pi^*(a|s) = arg\max_a Q^*(s,a) = arg\max_a \left(\max_\pi Q^\pi(s,a) \right). \tag{4}$$

B. Recovery Policy

A recovery policy is a specialized policy designed to ensure safety by guiding the agent away from unsafe or high-risk states [31]. When the agent is in a state where it is likely to violate constraints, the policy guides the agent back to the *safe set* to ensure safety during learning and execution. Recovery policy is operated by the belowed processes. First, the safety of a state-action pair (s, a) is evaluated using the safety critic $Q^{\pi}_{risk}(s, a)$, which estimates the probability of future constraint violations under the current policy π:

$$Q^{\pi}_{risk}(s,a) = \mathrm{E}_{\pi}[\sum_{\mathrm{t}=0}^{\infty} \gamma^{t}_{\mathrm{risk}} C(s_t, a_t)], \tag{5}$$

where $C(s_t, a_t)$ is the constraint cost function and $\gamma^{t}_{\mathrm{risk}}$ is the discount factor for the constraint cost. Second, we define the *safe set* (T^{π}_{safe}) and *recovery set* ($T^{\pi}_{recovery}$) as follows:

$$T^{\pi}_{safe} = \{(s,a) \mid Q^{\pi}_{risk}(s,a) \leq \varepsilon_{risk}\}, \tag{6}$$

$$T^{\pi}_{recovery} = \mathrm{S} \times \mathrm{A} \setminus T^{\pi}_{safe} \tag{7}$$

where $\varepsilon_{risk} \in [0,1]$ is a risk threshold. The sets $\mathrm{T}^{\pi}_{\mathrm{safe}}$ and $T^{\pi}_{recovery}$ refer to the state and action pair for safe and recovery, respectively, when the policy π is followed. The goal of the recovery policy is to create a policy where all state-action pairs are within the safe set, but to create a policy that recovers even if an agent is outside the safe set. Third, the agent uses a composite policy that combines the task policy π_{task} and the recovery policy $\pi_{recovery}$. Depending on the safety of the current state-action pair, the policy switches as follows:

$$a = \begin{cases} a_{task} & if \ \ (s, a_{task}) \in T^{\pi}_{safe} \\ a_{recovery} & if \ \ (s, a_{task}) \notin T^{\pi}_{safe} \end{cases}, \tag{8}$$

where $a_{task} \sim \pi_{task}(a|s)$ and $a_{recovery} \sim \pi_{recovery}(a|s)$ are the actions suggested by the task and recovery policy, respectively. If the agent is in the safe set, the task policy operates normally. On the contrary, if the agent is in the recovery set, the recovery policy takes over to guide the agent back to the safe set. Finally, the recovery policy $\pi_{recovery}$ is trained to minimize the safety critic $Q^{\pi}_{risk}(s, a)$, ensuring that the agent transitions back into the safe set:

$$\pi_{recovery} = arg \min_{\pi} \mathbb{E}_{(s,a) \in T^{\pi}_{recovery}} [Q^{\pi}_{risk}(s,a)]. \tag{9}$$

C. Control Barrier Functions

The recovery policy is a behavioral policy to achieve safety, and it does not guarantee safety. Therefore, it is necessary to mathematically define the safety constraints, which is called the Control Barrier Function (CBF). The CBF uses constraints to ensure the safety of the control system [12, 24], and how to design the control barriers depends on the system and the environment. To explain CBF, we first define a dynamic system as follows:

$$\dot{s} = f(s) + g(s)u(s), \tag{10}$$

where $s \in \mathbb{R}^{\mathrm{n}}$ and $u(s) \in \mathrm{U} \subseteq \mathbb{R}^{\mathrm{m}}$ are state and control input, respectively. The function $f(s)$ represents the intrinsic or natural behavior of the system when there is no external control input, and $g(s)$ describes how the control input $u(s)$ affects the state of the system. The functions $f(s)$ and $g(s)$ are locally Lipschitz.

A *safe set* is a subset of the state space of a system in which the system is guaranteed to operate safely. The safe set C is defined as follows:

$$C = \{s \in \mathbb{R}^{\mathrm{n}} : h(s) \geq 0\}, \tag{11}$$

where $h: \mathbb{R}^{\mathrm{n}} \to \mathbb{R}$ is a continuously differentiable scalar function defining the safe set C.

Using the definition of the safe set C, we define the boundary, the interior, and the exterior of the safe set as follows:

$$\partial C = \{s \in \mathbb{R}^{\mathrm{n}} : h(s) = 0\}, \tag{12}$$

$$\mathrm{Int}(C) = \{s \in \mathbb{R}^{\mathrm{n}} : h(s) > 0\}, \tag{13}$$

$$\mathbb{R}^{\mathrm{n}} \setminus C = \{s \in \mathbb{R}^{\mathrm{n}} : h(s) < 0\}. \tag{14}$$

To ensure safety, the state s should remain in the safe set C for all the time. This is referred to as *forward invariance* [32], which can be expressed as:

$$s(0) \in C \ \Rightarrow \ s(t) \in C, \quad \forall t \geq 0. \tag{15}$$

Proposition 1 (Safety via CBF): If the control input satisfies the CBF condition $\dot{h}(s,u) + \alpha(h(s)) \geq 0$, then the safe set C is *forward invariant*. That is, the robot state remains in the safe set C for all $t \geq 0$ if it starts in C.

The proposition 1 can be enforced by designing a control input u that satisfies the CBF condition:

$$\sup_{u \in \mathrm{U}} [L_f h(s) + L_g h(s) u] \geq -\alpha(h(s)), \tag{16}$$

where, $L_f h(s)$ and $L_g h(s)$ are the effect of the system's natural dynamics on $h(s)$ and the effect of the control input u on $h(s)$, respectively. The function $\alpha(h(s))$ is a K_{∞} function that adjusts the rate α at which the system approaches the safe set. Our main objective is finding appropriate control input u by maintaining CBF condition Eq. (16).

IV. Problem Formulation

The objective of the proposed method is to enable the robot to reach a goal safely and efficiently without collision. To achieve this objective, we first modify the standard MDP formulation in Eq. (1). Specifically, we introduce a constraint cost function $C(s) \to \{0,1\}$ that indicates whether a state is safe; a risk discount factor $\gamma_{risk} \in [0, 1]$ according to the constraint, and the negative reward $\mathrm{R}_{\mathrm{risk}}(s, a) \in (-\infty, 0)$. The modified MDP $\mathcal{M}'$ is defined as follows:

$$\mathcal{M}' = \langle \mathrm{S}, \mathrm{A}, \mathrm{P}, \mathrm{R}, \gamma, \mathrm{C}, \gamma_{\mathrm{risk}}, \mathrm{R}_{\mathrm{risk}} \rangle, \quad (17)$$

and the state of the robot s is defined as follows:

$$s = [d_{goal}, \theta_{goal}, \mathcal{O}], \quad (18)$$

where d_{goal} is the Euclidean distance from the center of the robot to the goal, and θ_{goal} is the angle to the goal with respect to the robot's orientation. The array $\mathcal{O}$ contains the distances and angles to neighboring obstacles:

$$\mathcal{O} = \{d_i^{obs}, \theta_i^{obs} | i = 1,2, \cdots, n\}, \quad (19)$$

where d_i^{obs} and θ_i^{obs} are the Euclidean distance and angle to the i-th obstacle, respectively. The time step t is omitted because the robot determines its action based solely on the current state. Our objective is to find an appropriate input $u(s)$ in Eq. (10) to ensure safety in the modified MDP.

V. The Definitions of Danger Levels

We define the robot's danger levels according to collision probability as follows:

- Safe level ($\mathcal{S}$): The robot is not in danger of collision and there are no obstacles around it. At the safe level, the robot can move freely.
- Risk level ($\mathcal{R}$): There is at least one obstacle near the robot with a potential collision. The robot should take a constrained action to avoid the collision at the risk level.
- Critical level ($\mathcal{C}$): The distance between the robot and the obstacles is too close; the risk of collision is very high. At the critical level, the robot must take an action aimed exclusively at avoiding collisions.

To define the boundaries of the danger level, we set separate thresholds to distinguish between risk and critical levels: $d_{\mathcal{R}}$ and $d_{\mathcal{C}}$. First, the threshold $d_{\mathcal{R}}$ is used to define the constraint triggers of the CBF. We vary $d_{\mathcal{R}}$ depending on the size and number of obstacles, the width of the environment, and the size of the robot, as constraint ranges can have a significant impact on performance. Especially in uncertain or dynamic environments, it is important to choose the appropriate range for constraints in CBF [33]. Second, the threshold $d_{\mathcal{C}}$ should be as small as possible to avoid the obstacle as follows:

$$d_{\mathcal{C}} = (d_{-v} + d_{\omega} + r), \quad (20)$$

$$d_{-v} = \frac{v_{max}^2}{2a_{decel}}, \quad (21)$$

$$d_{\omega} = \frac{\omega_{max} \cdot d_{wheels}}{2} \cdot \mathrm{T}_s, \quad (22)$$

where d_{-v} is the maximum braking distance based on linear velocity v_{max}, and d_{ω} is the maximum turning distance based on angular velocity. We define a_{decel} to be the deceleration of the robot, and d_{wheels} to be the distance between the two wheels. The T_s is the sampling time of the time step, and r is robot's radius.

The criteria for determining whether each danger level has been reached can be defined as determine functions using $d_{\mathcal{R}}$ and $d_{\mathcal{C}}$, as follows:

$$D_{\mathcal{R}}(s) = \max_{j \in 1,2,\ldots,n} \left(d_{\mathcal{R}} - d_j^{obs}\right), \quad (23)$$

$$D_{\mathcal{C}}(\mathrm{s}) = \max_{j \in 1,2,\ldots,n} \left(d_{\mathcal{C}} - d_j^{obs}\right). \quad (24)$$

where n is the number of obstacles. If the function is greater than or equal to zero, the corresponding danger level is considered to have been reached.

We introduce the danger level sets $\mathcal{S}^{set}$, $\mathcal{R}^{\mathrm{set}}$, $\mathcal{C}^{\mathrm{set}}$, which represent the sets of states corresponding to the safe, risky, and critical levels, respectively. The overall danger level set can be expressed as follows:

$$\mathcal{S}^{set} = \{s \in \mathbb{R}^{n+2} | D_{\mathcal{R}}(s) < 0\} \quad (25)$$

$$\mathcal{R}^{\mathrm{set}} = \{s \in \mathbb{R}^{n+2} | D_{\mathcal{R}}(s) \geq 0\} \cap \{s \in \mathbb{R}^{n+2} | D_{\mathcal{C}}(s) < 0\} \quad (26)$$

$$\mathcal{C}^{\mathrm{set}} = \{s \in \mathbb{R}^{n+2} | D_{\mathcal{C}}(s) \geq 0\} \quad (27)$$

where $\mathcal{S}^{set}$ is the set of states in which the robot maintains a safe distance from the obstacle. Similarly, $\mathcal{R}^{\mathrm{set}}$ is the set states when the robot is at a risk level, and $\mathcal{C}^{\mathrm{set}}$ is the set states when the robot is at a critical level. The reason why the dimension of the state is $\mathbb{R}^{n+2}$ is that the number of obstacles, the distance to the goal, and the angle are added as in Eqs. (18) and (19).

The desired controller using the danger level sets is as follows:

$$u(s) = u_{\mathcal{S}} \mathbb{I}_{\mathcal{S}^{set}}(\mathrm{s}) + u_{\mathcal{R}} \cdot \mathbb{I}_{\mathcal{R}^{set}}(s) + u_{\mathcal{C}} \cdot \mathbb{I}_{\mathcal{C}^{set}}(s) \quad (28)$$

where $u_{\mathcal{S}}$, $u_{\mathcal{R}}$, and $u_{\mathcal{C}}$ represent the control inputs for the safe, risky, and critical levels, respectively. The $\mathbb{I}_{\mathcal{S}^{set}}(\mathrm{s})$, $\mathbb{I}_{\mathcal{R}^{set}}(s)$, and $\mathbb{I}_{\mathcal{C}^{set}}(s)$ are indicator functions that determine whether the robot's state s belongs to each set of danger levels. The indicator function returns 1 if the robot is at that danger level and 0 otherwise. For example, if the robot is in the safe state $\mathcal{S}^{set}$, $u_{\mathcal{S}}$ is applied as the control input.

VI. Control Recovery and Barrier Function

A. Control Barrier Function

If the robot's state belongs to $\mathcal{R}^{set}$, the CBF is used to constrain the robot's control input. The CBF uses an *action filter* that eliminates unsafe actions from the candidate set and forms a filtered array of safe actions, denoted as $a^{filtered}$. The primary objective of the action filter is to evaluate the safety of each action defined during training and exclude those that violate the safety constraints.

Algorithm 1 shows the pseudo-code of the action filter. First, we heuristically set the scaling factor λ^f and threshold factor $\mathcal{T}$ to avoid filtering out safe actions by considering the robot's size, speed, and $d_{\mathcal{R}}$ (lines 1–2).

First, we generate the transition and rotation velocities $v_{\mathcal{R}}$ and $\omega_{\mathcal{R}}$, then we generate $O^{\mathcal{R}}$ which contains information about the obstacles contained in $\mathcal{R}^{set}$. The robot accumulates the calculated v and ω for relative distances and angles where obstacles are detected (lines 5–8). The accumulated $v_{\mathcal{R}}$ and $\omega_{\mathcal{R}}$ are multiplied by the scaling factor λ^f and added to the v_k and ω_k of the predefined action to generate v_{adj} and ω_{adj} (lines 10–11). In lines 12–16, the threshold $\mathcal{T}$ is multiplied by v_{adj} and ω_{adj} and compared with the values in the actions to determine whether each action is safe. By repeating the process for predefined actions (lines 9–17), the action filter returns the set of safe actions, $a^{filtered}$. The set $a^{filtered}$ has an array with the values indicate 1 for safe actions and 0 for risky actions, respectively.

Algorithm 1. Action Filter

1: Set $\lambda^f \in (0, 1]$ ▷ Scaling factor control adjustment
2: Set $\tau \in (0, 1]$ ▷ Scaling factor control adjustment
3: Initialize $v_{\mathcal{R}} \leftarrow 0, \omega_{\mathcal{R}} \leftarrow 0$
4: Initialize $O^{\mathcal{R}} \leftarrow \{(d_j^{obs}, \theta_j^{obs})|s \in \mathcal{R}^{set}\}$
5: **for** each $(\mathrm{d_j^{obs}}, \mathrm{\theta_j^{obs}}) \in \mathrm{O^{\mathcal{R}}}$ **do**
6: Calcuate $v_{\mathcal{R}} \leftarrow v_{\mathcal{R}} - cos(\theta)(d_{\mathcal{R}} - d_j^{obs})$
7: Calcuate $\omega_{\mathcal{R}} \leftarrow \omega_{\mathcal{R}} - sin(\theta)(d_{\mathcal{R}} - d_j^{obs})$
8: **end for**
9: **for** each $action\ a_k = (v_k, \omega_k)$ in the defined action **do**
10: Calcuate $v_{adj} \leftarrow v_k + \lambda^f \cdot v_{\mathcal{R}}$
11: Calcuate $\omega_{adj} \leftarrow \omega_k + \lambda^f \cdot \omega_{\mathcal{R}}$
12: **if** $|v_{adj}| < \tau \cdot |v_k|$ **or** $|\omega_{adj}| < \tau \cdot |\omega_k|$ **then**
13: Mark a_k as risky ($a_k \leftarrow 0$)
14: **else**
15: Mark a_k as safe ($a_k \leftarrow 1$)
16: **end if**
17: **end for**
18: **return** Set of safe actions $a^{filtered} = [a_1, a_2, \cdots, a_K]$

After the action filter eliminates unsafe actions, the remaining process in the CBF module involves weighting each candidate action by its selection probability, denoted as $pred_i$, and multiplying it by the filtered action vector $a^{filtered}$. Here, the probability $pred_i$ is associated with each candidate action generated during the selection of the RL-based action a^{NN}. The final action is selected as the one with the highest weighted value, denoted a^{CBF}, corresponding to the control input $u_{\mathcal{R}}$. Meanwhile, the RL-based action a^{NN} corresponds to the control input $u_{\mathcal{S}}$.

B. Recovery Function

The CBF is generally sufficient to ensure safety in mostly static environments. However, the CBF may fail if the sampling time T_s is too long or if obstacles move too quickly. In such cases, the CBF requires a wider constraint to maintain safety. While increasing this range can improve safety, it also makes the robot overly conservative, leading to a decrease in task efficiency due to its reluctance to take necessary risks.

To address this problem, we introduce a Recovery Function (RF) to improve the robot's safety, especially in high-risk situations. When the robot enters the critical danger level $\mathcal{C}$, the RF takes over control and generates a new control input to guide the robot towards safety. Specifically, the RF computes a recovery action a^{RF}, consisting of linear and angular velocity commands, designed to move the robot away from imminent danger. Unlike the CBF, which aims to maintain safety, the RF actively seeks to recover the robot to a safe state. Accordingly, the reference set used in the calculation of the RF is $\mathcal{R}^{set}$, not $\mathcal{C}^{\mathrm{set}}$, because the objective of the RF is not only to leave the critical area, but to return the robot completely to the safe area $\mathcal{S}$.

The pseudo-code of the recovery function is described in Algorithm 2. The goal of RF is to compute v and ω for s to be directed to $\mathcal{S}$. First, we set a scaling factor λ^r to account for the limitations of the robot's control input (line 1). We initialize $v_{\mathcal{C}}$ and $\omega_{\mathcal{C}}$ to zero, respectively (lines 2–3) and accumulate the computed v and ω for all detected obstacles (lines 4–7). Up to this point, it is almost identical to the action filter, with one caveat: the goal of RF is to recover to a safe state, which means that the information about obstacles used to compute v and ω also includes obstacles within the range of $s \in \mathcal{R}^{set}$. Finally, the recovery control $u_{\mathcal{C}}$, whose components are $v_{\mathcal{C}}$ and $\omega_{\mathcal{C}}$, is returned in line 8. For convenience, we call $u_{\mathcal{C}}$ as the action a^{RF}.

Algorithm 2. Recovery Function

1: Set $\lambda^f \in (0, 1]$ ▷ Scaling factor control adjustment
2: Initialize $v_C \leftarrow 0, \omega_C \leftarrow 0$
3: Initialize $O^{\mathcal{R}} \leftarrow \{(d_j^{obs}, \theta_j^{obs})|s \in \mathcal{R}^{set}\}$
4: **for** each $(d_j^{obs}, \theta_j^{obs}) \in O^{\mathcal{R}}$ **do**
5: Calcuate $v_C \leftarrow v_C - cos(\theta)(d_C - d_j^{obs})$
6: Calcuate $\omega_C \leftarrow \omega_C - sin(\theta)(d_C - d_j^{obs})$
7: **end for**
8: **return** $u_C = \lambda^r \cdot \begin{bmatrix} v_C \\ \omega_C \end{bmatrix}$

C. CRBF-Based SafeRL Controller

When an unsafe situation occurs, i.e., risky and critical, the CBF or RF selects an appropriate action to ensure safety. The CRBF framework enhances safety by integrating both CBF and RF mechanisms. The core function of the CRBF is to select the most appropriate action based on the current danger level, from among up to three candidate actions generated by the neural network and the CRBF modules.

Fig. 1 illustrates the possible correspondence between danger levels and selected actions in CRBF. The risky and critical areas, shown semi-transparently in Fig. 1, are the obstacle detection regions defined by Eqs. (23) and (24), respectively. When the robot's current danger level is $\mathcal{S}$(safe), the controller directly predicts and executes an action using the standard reinforcement learning policy. When the danger level is $\mathcal{R}$ (risky), the robot filters out unsafe actions using the action filter and executes one of the remaining valid actions. In cases like the Risky* example in Fig. 1, if no valid actions remain after filtering, the robot will stop and wait until the obstacle is no longer present. If the danger level is $\mathcal{C}$ (critical), the controller

generates and executes a recovery action designed to guide the robot back to the safe area $\mathcal{S}$. In the Critical* case shown in Fig. 1, if there are several obstacles in the critical and risky areas, the recovery action is calculated taking into account the obstacles in the adjacent area, ensuring a safe return to $\mathcal{S}$.

The pipeline of a CRBF-based controller is shown in Fig. 2. The first stage is learning by the neural network. Given a state, the neural network generates actions a^{NN} like a vanilla RL, and in the process, passes $pred_i$, the probability that each action used will be executed, to the CBF. The second stage is the CRBF. The CRBF is divided into the CBF and the RF. To avoid reducing computational efficiency by calculating actions that will never be used, the CRBF determines whether to use the CBF or the RF based on the danger level. Therefore, the CRBF will have zero to two actions. For simplicity, we will say that the CRBF always calculates both actions. When all actions are computed, the action selector chooses one of up to three actions that matches the current danger level, which is called a^{best}. The action a^{best} is passed to the environment to execute the action, and in the next state, we go back to the beginning of this paragraph and repeat the process. The behavior of the action selector is exactly the same as in Eq. (28); the action a^{best} is equal to the result of $u(s)$.

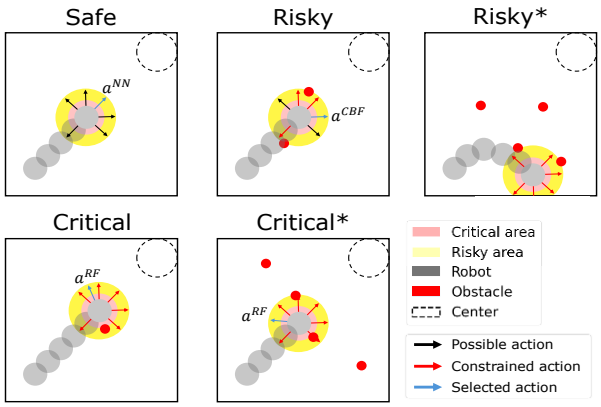

Fig. 1. Examples of actions in each situation when the robot used a CRBF-based SafeRL controller. These are the general responses of the controller in safe, risky, and critical situations, respectively. In a risky* situation, the robot stops until the obstacle is removed because the actions are constrained for all directions.

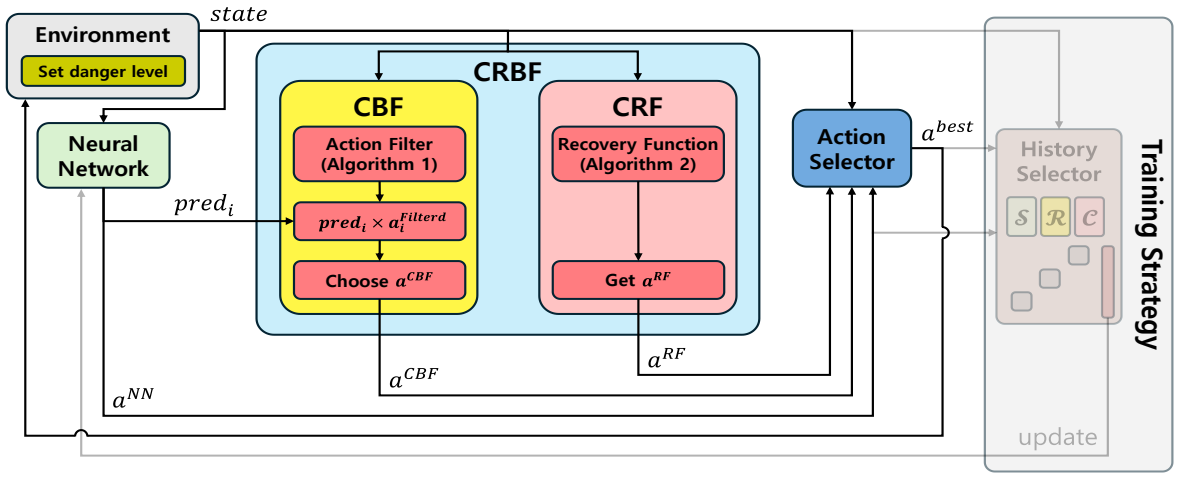

Fig. 2. CRBF-based SafeRL control pipeline. Neural network and CRBF generate state dependent actions. The actions in a CRBF come from two sources: one from the CBF and one from the CRF. The action selector selects one of the generated actions based on the danger level. The selected action is used as the control input.

The control inputs $u_{\mathcal{S}}$, $u_{\mathcal{R}}$, and $u_{\mathcal{C}}$ correspond to intermediate decisions made by the RL policy, the CBF-based controller, and the recovery function, respectively. Each control input ultimately produces a velocity command pair consisting of translational velocity v and rotational velocity ω. These values are applied directly to the robot's differential drive system. In the safe area, $u_{\mathcal{S}}$ is mapped from the RL policy output to ($v_{\mathcal{S}}$, $\omega_{\mathcal{S}}$). In the risky area, the CBF modifies this to ensure safety, producing ($v_{\mathcal{R}}$, $\omega_{\mathcal{R}}$). In the critical area, the recovery function generates emergency velocities ($v_{\mathcal{C}}$, $\omega_{\mathcal{C}}$) to move the robot away from obstacles. A control switching mechanism then selects the appropriate (v, ω) based on the current danger level, as shown in Algorithm 2.

Meanwhile, the training strategy in the right part of Fig. 2 is a simplified representation of the policy update part, which uses different policy update methods depending on the danger level. A detailed description of the policy update is continued in the next section.

VII. CRBF-Based SafeRL Training Strategy

There have already been several attempts to learn SafeRL, including SafeRL using CBF [31, 34]. However, we could not use the traditional method because we apply two constraints, CBF and RF, sequentially. To the best of our knowledge, there are few SafeRL training methods that use sequential constraints from a collision avoidance perspective to update policies. In this section, we describe the CRBF-based SafeRL training strategy.

A. Rewards and Penalties

Table I shows the rewards and penalties used for training. We call the reward received at the end of a single step R_{step}. We generate two penalties in addition to the

reward: P_{CBF} and P_{RF}. The penalty P_{CBF} is used when a^{NN} is filtered by the control barrier, and the penalty P_{RF} is given when the robot's danger level is $\mathcal{C}$.

TABLE I. REWARDS AND PENALTIES

Symbol	Description
R_{step}	Reward for each step
P_{CBF}	Barrier penalty
P_{RF}	Recovery penalty

B. Sequential Constraints Training Strategy

The CRBF-based SafeRL is a method for increasing safety by selecting one of three actions based on the danger level. One of the key features of SafeRL algorithms is that the action predicted by the neural network differs from the action actually taken. Accounting for this in policy updates is an important part of SafeRL's policy update process. However, none of the existing work has a policy update approach that is suitable for our proposed approach, and we propose a policy update strategy for CRBF-based SafeRL by combining and modifying existing approaches.

We use update policies to perform constrained actions as follows [35]:

- Unconstrained Prediction (UP): The UP is the same method as general reinforcement learning, where the neural network executes the predicted action without considering constraints.
- Constrained Prediction with Penalty (CPP): The CPP modifies the predicted action of the neural network to satisfy the constraints. The predicted action is used for training, but the changed action is actually executed to ensure a safe learning process with penalty.
- Constrained Prediction and Correction (CPC): The CPC closes the gap between the initial prediction and the corrected action by learning both the predicted and corrected actions. The CPC takes into account the cost of violating constraints by applying penalties to the rewards for the predicted action.

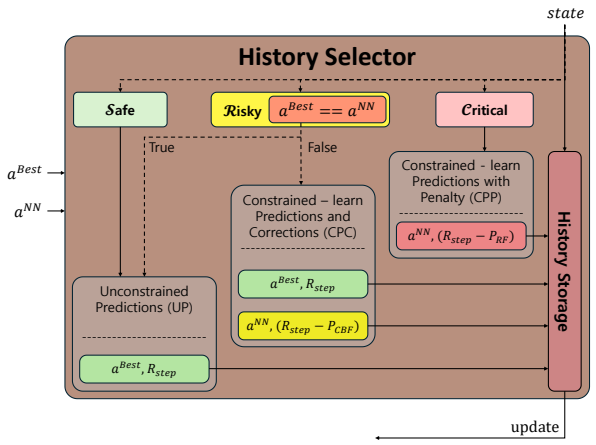

Fig. 3. History selector pipeline. The history selector chooses the policy update method for efficient learning when updating policies. Depending on the strategy selected, it stores the history storage. A dotted arrow indicates when the direction changes based on the selection.

We propose a learning strategy that combines the UP, CPP, and CPC approaches. Our learning strategy is represented by the history selector in Fig. 3. The history selector uses different policy update strategies when danger levels are $\mathcal{S}$, $\mathcal{R}$, and $\mathcal{C}$. The first choice of the history selector is to make a primary choice based on danger level. Here, if the danger level is $\mathcal{S}$, we use the UP strategy; if it is $\mathcal{R}$, we use the CPP strategy; but if it is $\mathcal{C}$, we need to make another choice. If the danger level is $\mathcal{C}$, determine whether the neural network's action is filtered by a control barrier. If it is filtered, use a CPC strategy. However, if the neural network's action is unfiltered, use the UP strategy as an agent would with $\mathcal{S}$. The actions and rewards selected in the history selector are combined with the state in the history storage for use in policy updates.

VIII. SIMULATIONS

A. Simulation Environment

For the simulation, we used the Webots simulator [36] on a computer with a Geforce RTX 3080 and Intel i7-12700 3.6 GHz. We used TurtleBot3 Burger as a two-wheeled robot with a 2D-LiDAR [37]. LiDAR offset calibration was performed due to the error between the center of the robot's rotation axis and the center of the LiDAR. The radius of the TurtleBot3 Burger is 105 mm, the distance between the two wheels 0.16 m, and the maximum transition and rotation velocities are 0.22 m/s and 2.84 rad/s, respectively. Although our simulations are conducted using TurtleBot3 in Webots, the CRBF framework is modular and does not rely on platform-specific kinematics or sensors. The policy and switching logic of the proposed method can be applied to any differential-drive robot, or to other robotic platforms that provide similar low-level velocity control interfaces.

The sampling time T_s is set to 0.05 seconds. In Eq. (20), we set $d_{\mathcal{C}}$ to 0.25 m. For comparison, $d_{\mathcal{R}}$ is set to small ($d_{\mathcal{R}} = 1.4d_{\mathcal{C}}$), medium ($d_{\mathcal{R}} = 1.6d_{\mathcal{C}}$), and large ($d_{\mathcal{R}} = 1.8d_{\mathcal{C}}$). Depending on which of the three $d_{\mathcal{R}}$ is applied, the proposed CRBF models were divided into $Ours_S$, $Ours_M$, and $Ours_L$, and the CBF models for comparison were divided into CBF_S, CBF_M, and CBF_L. Here, S, M, and L mean small, medium, and large, respectively. For the traditional comparison algorithms, we used APF [6], a widely used method, and ORCA [38]. The repulsive threshold distance of APF and the neighborhood distance of ORCA, which are the distances at which each algorithm starts to detect obstacles, were set equal to the medium $d_{\mathcal{R}}$. Table II shows the hyperparameters used for training. These hyperparameters were chosen to optimize the performance of the PPO algorithm during training.

TABLE II. HYPERPARAMETERS FOR TRAINING

Parameter	Value
Batch size	5
Action bound	1.0
Max step	500
Learning rate (actor)	5e−5
Learning rate (critic)	3e−5
Discount factor	0.99
Loss cliping	0.2
Entropy loss	1.5e−3

The simulation environment was 3 m × 3 m with walls of sufficient height around the perimeter, as shown in Fig. 4. The training environment consisted of 4 circular obstacles, two static and two dynamic. The maximum speed of the dynamic obstacle is 0.1 m/s. The test environment is set up as a static map with 6 circular static obstacles; a Mix_easy map with 2 circular static and 2 circular dynamic obstacles; a Mix_hard map with 2 circular static, 2 dynamic, and 2 square dynamic obstacles; a dynamic map with 6 circular dynamic obstacles; and a Dynamic_fast map with 5 relatively fast circular dynamic obstacles. All dynamic obstacles move at less than the maximum speed of 0.1 m/s and bounce off when they reach the goal, the wall, or other obstacles. The obstacles in the Dynamic_fast map have a maximum speed of 0.3 m/s. The robot starts at a random position near a wall on the map, facing a random direction. The robot's task is to reach the central goal while avoiding collisions. Each model was tested 300 times for each map.

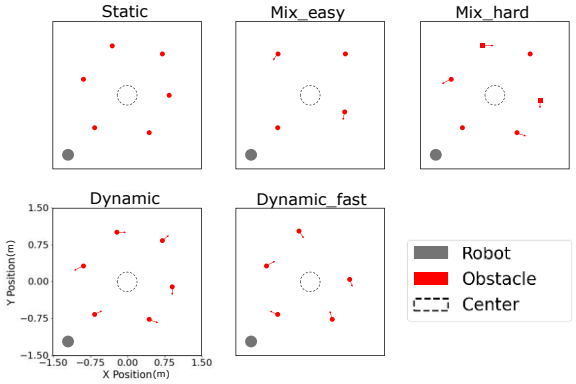

Fig. 4. Test environment for the simulation. The direction of dynamic obstacles is indicated by the red arrow.

During training, we applied domain randomization to achieve model generalization by randomly varying different variables in the simulated environment. Domain randomization is known as one of the effective approaches in Sim2Real, a research area that aims to reduce the difference between simulation and real environments [39].

The domain randomization is as follows:

1) We randomize the initial position and initial orientation of the robot.
2) We randomize the speed and initial orientation of each obstacle.
3) We apply Gaussian noise of up to 5 mm to the LiDAR sensor.
4) We apply Gaussian noise of up to 15% to the sampling time per time step.

B. Simulation Results

Table III shows the results of the collision avoidance simulations. For each model, we recorded the number of successful trials, collisions, and instances where the agent reached the maximum allowed steps. We defined a failure as a collision or reaching the maximum step. For the static map, all models achieved a collision rate of 3% or less when considering only the number of collisions, except when exceeding the max step. Except for the dynamic_fast map with obstacles moving at untrained speeds, all of our models achieved a collision rate of less than 1.2%. With the additional exception of $Ours_L$, which has the largest $d_{\mathcal{R}}$, our models maintained a success rate of above 97% in all cases. Meanwhile, in the dynamic_fast map, the $Ours_L$ model had the highest success rate.

TABLE III. SIMULATION RESULTS FOR COLLISION AVOIDANCE

Map	Model	# of success	# of collision	Success rate	# max step
Static	$Ours_S$	300	0	100.0 %	0
	$Ours_M$	291	0	97.0 %	9
	$Ours_L$	173	0	57.7 %	127
	RF	201	0	67.0 %	99
	CBF_S	291	9	97.0 %	0
	CBF_M	245	6	81.7 %	49
	CBF_L	163	0	54.3 %	137
	APF	300	0	100.0 %	0
	ORCA	300	0	100.0 %	0
Mix_easy	$Ours_S$	298	0	99.3 %	2
	$Ours_M$	296	0	98.7 %	4
	$Ours_L$	298	0	98.7 %	4
	RF	287	1	95.7 %	12
	CBF_S	286	14	95.3 %	0
	CBF_M	285	10	95.0 %	5
	CBF_L	285	2	95.0 %	13
	APF	299	1	99.7 %	0
	ORCA	297	3	99.0 %	0
Mix_hard	$Ours_S$	296	0	98.7 %	4
	$Ours_M$	293	1	97.7 %	6
	$Ours_L$	234	1	78.0 %	65
	RF	244	3	81.3 %	53
	CBF_S	261	34	87.0 %	5
	CBF_M	239	50	79.7 %	11
	CBF_L	221	58	73.7 %	21
	APF	298	2	99.3 %	0
	ORCA	294	6	98.0 %	0
Dynamic	$Ours_S$	298	0	99.3 %	8
	$Ours_M$	299	0	99.7 %	1
	$Ours_L$	295	0	98.3 %	5
	RF	285	2	95.0 %	13
	CBF_S	250	50	83.3 %	0
	CBF_M	229	69	76.3 %	2
	CBF_L	233	33	77.7 %	34
	APF	599	1	99.7 %	0
	ORCA	288	12	96.0 %	0
Dynamic _fast	$Ours_S$	265	35	88.3 %	0
	$Ours_M$	282	18	94.0 %	0
	$Ours_L$	287	13	95.7 %	0
	RF	224	76	76.7 %	0
	CBF_S	207	93	69.0 %	0
	CBF_M	211	89	70.3 %	0
	CBF_L	191	107	63.7 %	2
	APF	207	93	69.0 %	0
	ORCA	222	78	74.0 %	0

Fig. 5 shows the success rate for each of the collision avoidance models. All learning-based models have a high success rate on the mix_easy map because it is very similar

to the training environment. The difference in success rate compared to traditonal collision avoidance algorithms is most noticeable on the dynamic_fast map with obstacles that are faster than the robot. In the dynamic_fast map, all our models had a significantly higher success rate than the traditional collision avoidance models. Models using RF only achieved high success rates on the dynamic maps.

The trajectory of a robot avoiding collisions in the dynamic_fast map is shown in Fig. 6. In the situation shown in the trajectory, CBF_S and APF collided with the obstacle. In the case of CBF_S, the CBF moved the robot to a safe place. At the same time, however, an obstacle moved into the robot's path, causing it to collide. The APF was not able to handle situations where an obstacle blocked the path and another obstacle was next to it. The other models moved well to their goal without colliding with any obstacles, and the model that arrived in the fastest time is ORCA. In Fig. 6, the areas that need to detect obstacles to set the danger level for each model are set to the same as used in Fig. 1.

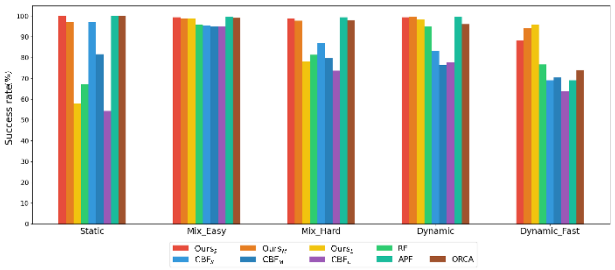

Fig. 5. Collision avoidance success rate for each collision avoidance model in simulations.

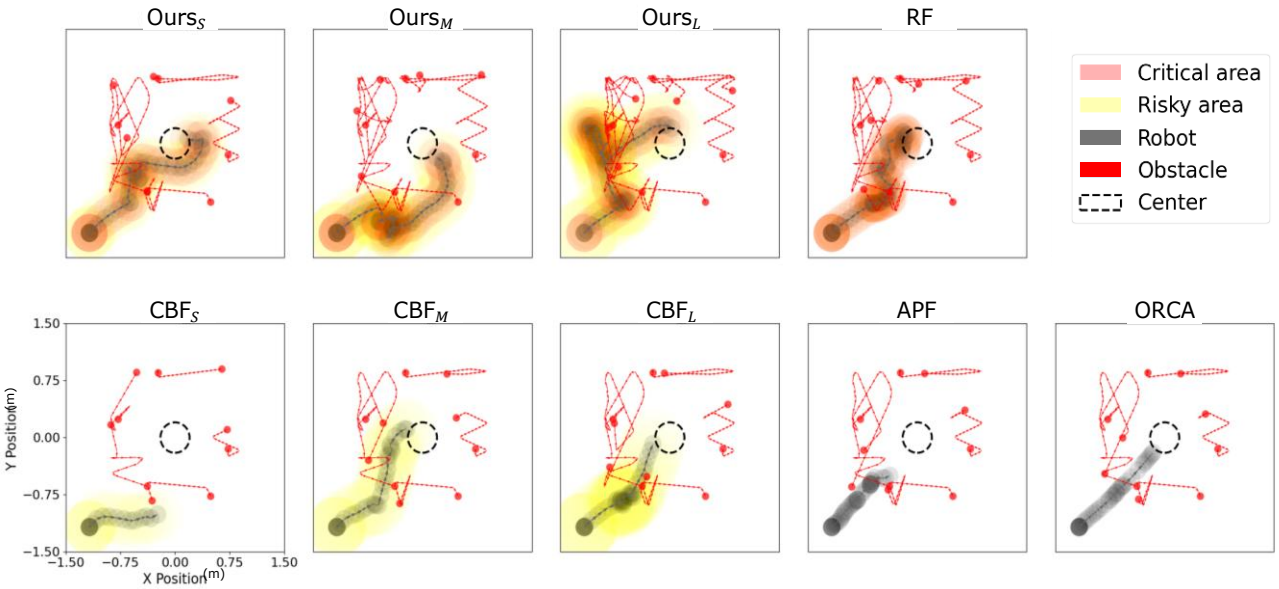

Fig. 6. The trajectories of each model and the obstacles in the first episode of the dynamic_fast map. The yellow and red translucent areas are where obstacles are detected to set the danger level for each model. The start and end points of the obstacles are represented by red dots, and the paths of the obstacles are represented by red arrows.

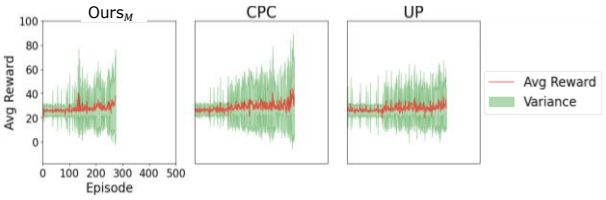

Fig. 7. Average reward and variance per episode during the training.

To demonstrate our proposed training strategy, we performed CRBF-based collision avoidance training using a different strategy. We set the learning criterion to $Ours_M$. Two additional models were trained: a model using the CPC method and a model using only the UP method. Fig. 7 shows the reward history during training for these three models. The criterion for the end of training is when the success rate is at least 95% for the mix_easy map and 70% for the mix_hard map. Since the generalization performance of the UP method was too low, we set the training termination criterion for the UP method when the success rate is at least 95% for the mix_easy map and at least 50% for all other maps.

The training time for our method, the CPC method, and the UP method were 3hr 44min, 5hr 17min, and 6hr 1min, respectively. The CPC method required 12.2% less training time than the UP method and achieved higher collision avoidance performance than the UP method, as shown in Table IV. Meanwhile, the $Ours_M$ method required 29.3% less training time than the CPC method, with little performance difference in performance between the $Ours_M$ and CPC methods.

From the results so far, while we observe consistent performance gains in success rate, we acknowledge that the current results do not include hypothesis testing or confidence intervals. In future work, we plan to repeat the experiments with a larger number of trials and apply statistical validation methods such as confidence intervals or t-tests to ensure the robustness of the results.

TABLE IV. SIMULATION RESULTS FOR DIFFERENT TRAINING METHODS

Map	Model	# of success	# of collision	Success rate	# max step
Static	Ours	291	0	97.0 %	9
	CPC	288	0	96.0 %	12
	UP	195	0	65.0 %	105
Mix_easy	Ours	296	0	98.7 %	4
	CPC	298	0	99.3 %	2
	UP	294	0	98.0 %	6
Mix_hard	Ours	293	1	97.7 %	6
	CPC	293	3	97.7 %	4
	UP	174	0	58.0 %	126
Dynamic	Ours	299	0	99.7 %	1
	CPC	298	1	99.3 %	1
	UP	222	6	74.0 %	72
Dyn_fast	Ours	282	18	94.0 %	0
	CPC	275	21	91.7 %	4
	UP	256	41	85.3 %	3

IX. PRACTICAL EXPERIMENTS

A. Environment

The TurtleBot3-Waffle, a robot similar to the one used in the simulation but of a different size, was used as the robot for the real-world experiment. Similarly, the LiDAR offset calibration was performed due to the error between the center of the TurtleBot3's rotation axis and the centre of the LiDAR. The radius of the TurtleBot3 waffle is 220 mm, the distance between the two wheels is 0.287 m, and the maximum translational and rotational velocities are 0.26 m/s and 1.82 rad/s, respectively. We set d_c to 0.40 m in Eq. (20). To localize a robot and obstacles, we used the Vicon robot tracking system [40], which can measure the robot's position with an accuracy of 2 mm. While the physical experiments are performed in a motion capture-based environment, the CRBF is independent of any particular localization method and can be applied to systems with on-board perception and sensor fusion.

We used ROS1 (Robot Operating System) to control the robot. We set up two similar test environments, as shown in Fig. 8. Both environments share two static obstacles in common, one at the top and one at the bottom, but differ in their dynamic obstacles. The first environment is a robot map with a dynamic obstacle that continuously patrols a path diagonally across the center from the bottom right to the top left. The dynamic obstacle has a speed of 0.26 m/s, the same as the robot. The second environment is a human map with a human patrolling the same path as the obstacle in the first environment. The human moves at a speed similar to that of the agent robot. The robot starts at a random location near a wall on the map, facing in a random direction.

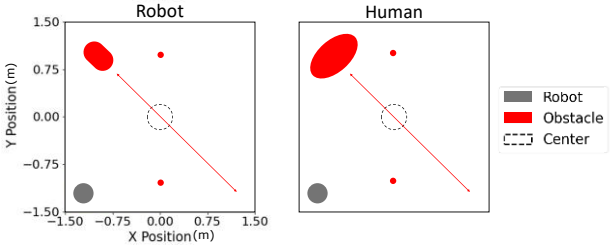

Fig. 8. Real-world test environment. The patrol direction of dynamic obstacles is indicated by the red arrow.

B. Results

Fig. 9 shows the collision avoidance success rate for each model. In the real world, the models showed sufficient time efficiency without requiring a specified maximum step. In the robot map, all models had a collision avoidance success rate above 85%. In the human map, the difference in success rates between our models and the CBF models is more pronounced. In the case of humans, the motion is not constant, but the proposed method has shown an excellent ability to cope with sudden changes in motion.

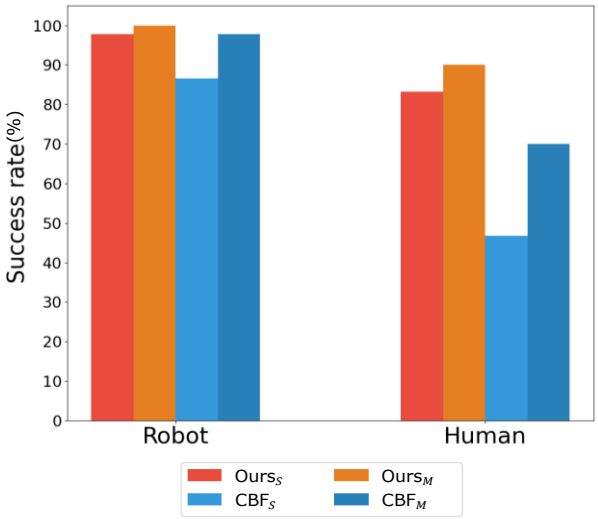

Fig. 9. Collision avoidance success rate for each model in robot and human maps.

Fig. 10 shows the trajectory when a dynamic obstacle is implemented with a robot. Since the obstacle has the same speed as the robot, the robot and the obstacle will always meet around the goal if the robot goes straight to the goal. The CBF_S method collided just before reaching the goal. The $Ours_S$ method, similar to the CBF_S method, hit the obstacle just before reaching the goal, but did not collide. The $Ours_M$ and CBF_M methods had a trajectory that avoided the obstacle and went around it.

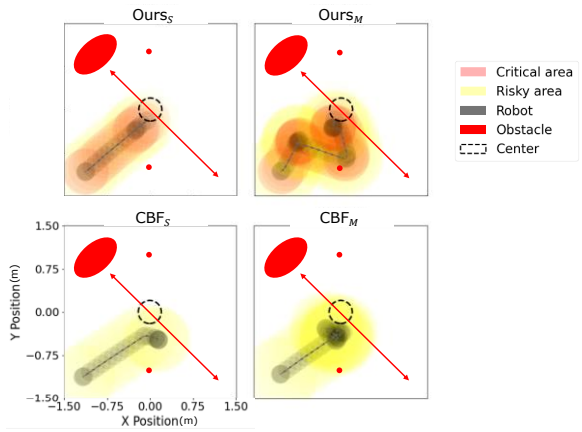

Fig. 10. The trajectories of each model in the first episode of the robot map. The dynamic obstacle is patrolled in the direction of the red arrows.

Fig. 11 shows the trajectory of an experiment performed on the human map. As the people appearing as obstacles

move at a similar speed to the robot, the robot will always encounter obstacles close to the goal if the robot goes straight to the goal. Again, only the CBF_S method collided with the obstacle. Like the other models, the CBF_S method took actions to avoid human obstacles, but not enough.

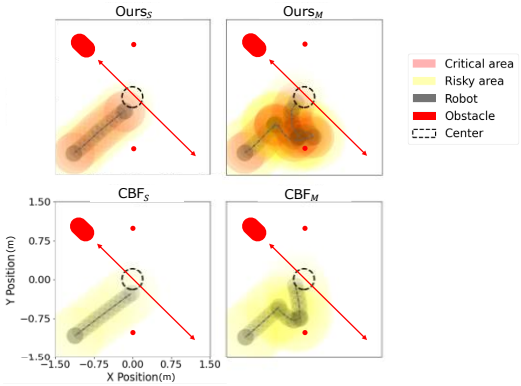

Fig. 11. The trajectories of each model in the first episode of the human map. The human patrolled in the direction of the arrows.

X. Discussion

The proposed SafeRL framework introduces a structured switching mechanism between RL, CBF, and RF policies that prioritizes safety while allowing efficient navigation to goal positions in dynamic environments. This hybrid control strategy effectively addresses one of the key limitations of conventional reinforcement learning, i.e., its inability to guarantee safety during both training and execution, without sacrificing the adaptability that makes RL attractive in dynamic and uncertain scenarios. By preserving the reactive and flexible nature of RL while embedding formal safety and recovery layers, the proposed method provides a balanced solution that improves both safety and task performance in real-world applications.

Although our framework integrates RL with CBF and RF modules to achieve both motion planning and safety, a full control-theoretic proof that formally guarantees convergence to the goal while maintaining safety is beyond the scope of this paper. However, the use of CBF ensures forward invariance of the defined safe set under certain conditions, and the recovery mechanism provides a fallback when standard control fails to maintain safety. We believe that these components provide the basis for future theoretical analysis.

While the proposed CRBF framework shows promising empirical performance in both simulation and real-world experiments, we acknowledge that formal guarantees of safety and task efficiency under model uncertainty are not yet established. Although random noise has been introduced into the sensor observations and dynamics to partially account for perception and control errors in simulation experiments, it is limited in assessing the robustness of the method under real-world uncertainty. The current method relies on a deterministic perception and control pipeline, which may limit its performance under severe sensor noise, dynamic uncertainties, or adversarial disturbances. As future work, we plan to extend the framework by incorporating robust control techniques or probabilistic safety verification, e.g. using Gaussian processes or reachability analysis, to provide formal guarantees in stochastic environments.

Furthermore, our current evaluation focuses on standard SafeRL and classical baselines, but our framework differs from recent hybrid methods by introducing a danger-aware hierarchical controller that dynamically switches between RL, CBF, and RF policies. Future work will extend this comparison to include more recent methods such as RecoveryRL [41] and robust CBF-RL [42].

XI. Conclusion

This paper proposes a CRBF-based SafeRL model for collision avoidance and presents a training method. The CRBF improves the model's collision avoidance performance over traditional methods by enforcing sequential constraints as elements of SafeRL. In addition, our proposed CRBF training method showed faster training times and higher generalization performance than existing methods. The proposed CRBF does not explicitly account for uncertainty or dynamics in the environment. However, the experiments showed that adding RF with sequential constraints reduces conflicts due to uncertainty that may occur when using CBF alone. In future work, we will develop a crash-resistant SafeRL algorithm that incorporates uncertainty in the environment.

Nomenclature

Symbol	Description
s	State of the robot
a	Action taken by the robot
π	Policy function
r(s, a)	Reward function
γ	Discount factor
P(s'|s, a)	State transition probability
Q(s, a)	State-action value function
$\mathcal{S}$	State space
$\mathcal{A}$	Action space
u	Control input
$u_S, u_\mathcal{R}, u_\mathcal{C}$	Control input in the safe, risky, and critical areas
v, ω	Translational and rotational velocities
d_i^{obs}	Euclidean distance to the i-th obstacle
θ_i^{obs}	Angle to the i-th obstacle
d_{goal}	Distance to the goal
θ_{goal}	Angle to the goal
$a^{filtered}$	Set of safe actions filtered by CBF
$\mathcal{S}^{set}, \mathcal{R}^{set}, \mathcal{C}^{set}$	State set of safe, risk, and critical level

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Conceptualization, Gyuyong Hwang and Gyuho Eoh; methodology, Gyuyong Hwang; validation, Gyuyong Hwang; writing—original draft preparation, Gyuyong Hwang; writing—review and editing, Gyuho Eoh; project administration, Gyuho Eoh; funding acquisition, Gyuho Eoh. All authors have read and agreed to the published version of the manuscript.

FUNDING

This research was supported by a grant (D2403003) from Gyeonggi Technology Development Program funded by Gyeonggi Province. This work was also supported by the GRRC program of Gyeonggi province [GRRC TUKorea2023-B03, Development of an intelligent inspection system and an autonomous navigation system for the transportation of multi-material parts].

REFERENCES

[1] C. H. R. Everett, "Survey of collision avoidance and ranging sensors for mobile robots," *Robotics and Autonomous Systems*, vol. 5, no. 1, pp. 5–67, May 1989. https://doi.org/10.1016/0921-8890(89)90041-9

[2] S. Haddadi, A. De Luca, and A. Albu-Schaffer, "Robot collisions: A survey on detection, isolation, and identification," *IEEE Transactions on Robotics*, vol. 33, no. 6, pp. 1292–1312, Dec. 2017. https://doi.org/10.1109/TRO.2017.2723903

[3] C. W. Kim and J. M. A. Tanchoco, "Conflict-free shortest time bidirectional AGV routing," *International Journal of Production Research*, vol. 29, no. 12, pp. 2377–2391, Dec. 1991. https://doi.org/10.1080/00207549108948090

[4] G. Leitmann and J. Skowronski, "Avoidance control," *Journal of Optimization Theory and Applications*, vol. 23, no. 4, pp. 581–591, Dec. 1977. https://doi.org/10.1007/BF00933298

[5] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, Mar. 1986. https://doi.org/10.1177/027836498600500106

[6] A. Chakravarthy and D. Ghose, "Obstacle avoidance in a dynamic environment: A collision cone approach," *IEEE Transactions on Systems, Man, and Cybernetics: Part A: Systems and Humans*, vol. 28, no. 5, pp. 562–574, 1998. https://doi.org/10.1109/3468.709600

[7] R. S. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, Cambridge, Ma; London: The Mit Press, 2018. https://doi.org/10.1017/S0263574799271172

[8] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-aware reinforcement learning for collision avoidance," arXiv preprint, arXiv:1702.01182, 2017. https://doi.org/10.48550/arXiv.1702.01182

[9] W. R. Clements, B.-M. Robaglia, B. Van Delft, R. B. Slaoui, and S. Toth, "Estimating risk and uncertainty in deep reinforcement learning", in *Proc. ICML Workshop Uncertainty Robustness Deep Learning*, 2020. https://doi.org/10.48550/arXiv.1905.09638

[10] B. Charpentier, R. Senanayake, M. Kochenderfer, and S. Gunnemann, "Disentangling epistemic and aleatoric uncertainty in reinforcement learning", arXiv preprint, arXiv:2206.01558, 2022. https://doi.org/10.48550/arXiv.2206.01558

[11] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proc. the AAAI Conference on Artificial Intelligence*, Apr. 2018, vol. 32, no. 1. https://doi.org/10.1609/aaai.v32i1.11797

[12] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *Proc. 18th European Control Conference (ECC)*, Jun. 2019. https://doi.org/10.23919/ECC.2019.8796030

[13] P. Wieland and F. Allgöwer, "Constructive safety using control barrier functions," in *Proc. IFAC*, vol. 40, no. 12, pp. 462–467, Jan. 2007. https://doi.org/10.3182/20070822-3-ZA-2920.00076

[14] 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. https://doi.org/10.1109/TAC.2016.2638961

[15] I. Tezuka and H. Nakamura, "Strict zeroing control barrier function for continuous safety assist control," *IEEE Control Systems Letters*, vol. 6, pp. 2108–2113, 2022. https://doi.org/10.1109/LCSYS.2021.3138526

[16] Y. Hu, J. Fu, and G. Wen, "Safe reinforcement learning for model-reference trajectory tracking of uncertain autonomous vehicles with model-based acceleration," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 3, pp. 2332–2344, Mar. 2023. https://doi.org/10.1109/TIV.2022.3233592

[17] D. Wang, T. Fan, T. Han, and J. Pan, "A two-stage reinforcement learning approach for multi-UAV collision avoidance under imperfect sensing," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3098–3105, Apr. 2020. https://doi.org/10.1109/LRA.2020.2974648

[18] J. Woo and N. Kim, "Collision avoidance for an unmanned surface vehicle using deep reinforcement learning," *Ocean Engineering*, vol. 199, 107001, Mar. 2020. https://doi.org/10.1016/j.oceaneng.2020.107001

[19] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, May 01, 2018. https://doi.org/10.1109/ICRA.2018.8461113

[20] J. Liang, U. Patel, A. J. Sathyamoorthy and D. Manocha, "Realtime collision avoidance for mobile robots in dense crowds using implicit multi-sensor fusion and deep reinforcement learning", arXiv preprint, arXiv:2004.03089, 2020. https://doi.org/10.48550/arXiv.2004.03089

[21] M. Everett, Y. F. Chen, and J. P. How, "Collision avoidance in pedestrianrich environments with deep reinforcement learning," *IEEE Access*, vol. 9, pp. 10357–10377, 2021. https://doi.org/10.1109/ACCESS.2021.3050338

[22] J. Roghair, A. Niaraki, K. Ko, and A. Jannesari, "A vision based deep reinforcement learning algorithm for UAV obstacle avoidance," in *Proc. of SAI Intelligent Systems Conference*. Springer, 2021, pp. 115–128. https://doi.org/10.1007/978-3-030-82193-7_8

[23] B. Lütjens, M. Everett, and J. P. How, "Safe reinforcement learning with model uncertainty estimates," in *Proc. IEEE Internati onal Conference on Robotics and Automation (ICRA)*, May 2019, pp. 8662–8668. https://doi.org/10.1109/ICRA.2019.8793611

[24] M. Srouji, H. Thomas, Y.-H.-H. Tsai, A. Farhadi and J. Zhang, "SAFER: Safe collision avoidance using focused and efficient trajectory search with reinforcement learning," in *Proc. IEEE 19th International Conference Automation Science Engineering (CASE)*, pp. 1–8, Aug. 2023. https://doi.org/10.1109/CASE56687.2023.10260402

[25] Z.-Q. Zhou *et al.*, "A safe reinforcement learning approach for autonomous navigation of mobile robots in dynamic environments," in *Proc. CAAI Transactions on Intelligence Technology*, Oct. 2023, pp. 1–16. https://doi.org/10.1049/cit2.12269

[26] Z. Marvi and B. Kiumarsi, "Safe reinforcement learning: A control barrier function optimization approach," *International Journal of Robust and Nonlinear Control*, vol. 31, no. 6, pp. 1923–1940, Aug. 2020. https://doi.org/10.1002/rnc.5132

[27] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *Proc. 18th European Control Conference (ECC)*, Jun. 2019. https://doi.org/10.23919/ECC.2019.8796030

[28] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proc. the AAAI Conference on Artificial Intelligence*, Jul. 2019, vol. 33, pp. 3387–3395. https://doi.org/10.1609/aaai.v33i01.33013387

[29] Z. Cai, H. Cao, W. Lu, L. Zhang, and H. Xiong, "Safe multi-agent reinforcement learning through decentralized multiple control barrier functions," arXiv preprint, arXiv:2103.12553, 2021. https://doi.org/10.48550/arXiv.2103.12553

[30] Y. Emam, P. Glotfelter, Z. Kira, and M. Egerstedt, "Safe reinforcement learning using robust control barrier functions," arXiv preprint, arXiv:2110.05415, 2021. https://doi.org/10.48550/arXiv.2110.05415

[31] A. Anand, K. Seel, V. Gjærum, A. Håkansson, H. Robinson, and A. Saad, "Safe learning for control using control Lyapunov functions and control barrier functions: A review," *Procedia Computer Science*, vol. 192, pp. 3987–3997, 2021. https://doi.org/10.1016/j.procs.2021.09.173

[32] A. D. AMES *et al.*, "Control barrier functions: Theory and applications," In *Proc. 18th European Control Conference (ECC)*, June 2019, pp. 3420–3431. https://doi.org/10.23919/ECC.2019.8796030

[33] W. Xiao, Calin Belta, and C. G. Cassandras, "Adaptive control barrier functions," *IEEE Transactions on Automatic Control*, vol. 67, no. 5, pp. 2267–2281, May 2022. https://doi.org/10.1109/TAC.2021.3074895

[34] M. Guerrier, H. Fouad, and G. Beltrame, "Learning control barrier functions and their application in reinforcement learning: A survey", arXiv preprint, arXiv:2404.16879, 2024. https://doi.org/10.48550/arXiv.2404.16879

[35] T. H. Pham, G. De Magistris, and R. Tachibana, "OptLayer-practical constrained optimization for deep reinforcement learning in the real world," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 6236–6243. https://doi.org/10.1109/ICRA.2018.8460547

[36] O. Michel, "Cyberbotics Ltd. WebotsTM: Professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, 5, Mar. 2004. https://doi.org/10.5772/5618

[37] R. Amsters and P. Slaets, "Turtlebot 3 as a robotics education platform," in *Proc. International Conference Robotics Eduation. (RiE)*. Cham, Switzerland: Springer, 2020, pp. 170–181. https://doi.org/10.1007/978-3-030-26945-6_16

[38] J. v. den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Proc. Robotics Research*, Berlin, Germany: Springer, 2011, pp. 3–19. https://doi.org/10.1007/10.1007/978-3-642-19457-3_1

[39] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, pp. 23–30, Sep. 2017. https://doi.org/10.1109/IROS.2017.8202133

[40] P. Merriaux, Y. Dupuis, R. Boutteau, P. Vasseur, and X. Savatier, "A study of Vicon system positioning performance," *Sensors*, vol. 17, no. 7, 1591, Jul. 2017. https://doi.org/10.3390/s17071591

[41] B. Thananjeyan *et al.*, "Recovery RL: Safe reinforcement learning with learned recovery zones," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4915–4922, Jul. 2021. https://doi.org/10.1109/LRA.2021.3070252

[42] S. Edvards and P. Ögren. "Using reinforcement learning to create control barrier functions for explicit risk mitigation in adversarial environments," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Oct. 2021, pp. 10734–10740. https://doi.org/10.1109/ICRA48506.2021.9561853