Model-free Neural Lyapunov Control for Safe Robot Navigation

Zikang Xiong, Joe Eappen, Ahmed H. Qureshi, and Suresh Jagannathan

***Abstract*— Model-free Deep Reinforcement Learning (DRL) controllers have demonstrated promising results on various challenging non-linear control tasks. While a model-free DRL algorithm can solve unknown dynamics and high-dimensional problems, it lacks safety assurance. Although safety constraints can be encoded as part of a reward function, there still exists a large gap between an RL controller trained with this modified reward and a safe controller. In contrast, instead of implicitly encoding safety constraints with rewards, we explicitly co-learn a Twin Neural Lyapunov Function (TNLF) with the control policy in the DRL training loop and use the learned TNLF to build a runtime monitor. Combined with the path generated from a planner, the monitor chooses appropriate waypoints that guide the learned controller to provide collision-free control trajectories. Our approach inherits the scalability advantages from DRL while enhancing safety guarantees. Our experimental evaluation demonstrates the effectiveness of our approach compared to DRL with augmented rewards and constrained DRL methods over a range of high-dimensional safety-sensitive navigation tasks.**

I. Introduction

Conditioning the goal of a controller with waypoints generated by a planner is a natural approach to combine planning and control [1]. A low-level controller guides a robot to follow a path generated by a high-level planner and finally navigates a robot to achieve the desired goal. However, such a decomposition must also take into account the possibility that the controller may not exactly follow the planned path. For example, when an autonomous vehicle needs to make a U-turn, the planned path might be too sharp to follow since the planner is usually agnostic of the underlying controller's capabilities. This raises safety concerns - even if the planner can generate safe plans, it is not always the case that the controller can follow the given path.

Solving kinodynamic constraints [2] addresses the inconsistency problem between planning and control of a robot. Recently, the application of DRL in this domain has provided a scalable solution for addressing kinodynamic constraints. [3], [4], [5] train DRL local controllers to reach a waypoint while also avoiding collisions. Since these local controllers can avoid collisions, the safety of these controllers following the planned paths is further improved. These controllers achieve obstacle avoidance by formulating collisions as penalty items in the reward function. Notably, since the avoidance capability of these controllers is only implicitly encoded as part of the reward function, the actual avoidance capability of these controllers remains a question. Moreover, it is generally hard to decide the proportion of these penalty items. When this proportion is too large, the controller is likely to learn to stay in the initial position since this is always safe and can receive a better reward compared with exploring other regions of the space, potentially leading to collisions. If it is too small, the controller will likely ignore these collisions and only focus on achieving the goal. This phenomenon, known as reward hacking, often makes training DRL policy extremely challenging in high-dimensional spaces.

Authors affiliate to Computer Science Department, Purdue University, IN 47906, USA. (E-mail: xiong84@purdue.edu; jeappen@purdue.edu; ahqureshi@purdue.edu; suresh@cs.purdue.edu)

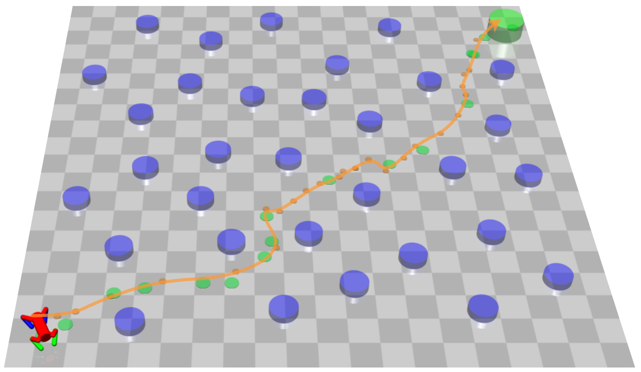


Fig. 1: Level-2 Quadruped Navigation Task. Blue cylinders represent hazardous zones, and the small green dots represent the waypoints generated by RRT. The final goal is the large green cylinder. We navigate the quadruped robot to achieve the goal while avoiding any hazardous zones.

The key idea of our approach is to explicitly place a sequence of robot reachability boundaries around a given plan. As long as we can ensure that no obstacle appears in the reachability boundary, we can provide enhanced safety guarantees when following a given plan. Lyapunov function and its region of attraction are widely used to build reachability boundaries. Combined with Neural Lyapunov Functions, (NLF) [6], [7], [8] which has been extensively studied, this approach has the potential to scale such reachability analyses to high-dimensional problems. However, effectively learning and analyzing the NLF is still an open question. In this paper, we firstly propose a co-learning algorithm to build the controller and the NLF in one training loop, which benefits both the controller and NLF. Furthermore, we analyze the NLF with a computationally effective approach. This allows our algorithm to be effectively deployed to realize safety-sensitive navigation tasks.

We evaluate our approach on the Safety Gym suite [9] in highly cluttered environments with three levels of obstacles complexity and each with four complex dynamical systems, named *sweeping* (similar to rumba), *point* (a circular rigid-body), *car*, and a 58 DOF *quadruped* (Fig. 1). These environments are challenging for traditional DRL methods, even without safety constraints. However, this paper demonstrates that our approach can scale well to these high-dimensional, cluttered navigation tasks while explicitly incorporating safety constraints. Our framework effectively co-learns a TNLF in the DRL loop and leverages the learned TNLF in a computationally tractable manner.

Our main contributions are as follows:

1) We propose a model-free Lyapunov method that can provide significant safety enhancement for high-dimensional safety-sensitive robot navigation problems with raw sensor observations.
2) We co-learn a TNLF and controller in a manner that improves controller performance and yields a high-quality Lyapunov function.
3) We use the learned TNLF to build a computationally effective runtime monitor for heuristically guiding robots under safety constraints at runtime.
4) We demonstrate that the combined planner and NLF can consistently reach the goal state in challenging safety-sensitive navigation tasks while also providing significantly fewer safety violations and a higher reach rate than modern constrained RL methods.

II. Background

A. Reinforcement Learning

Reinforcement learning (RL) generates an optimal controller by interacting with an environment given a scalar reward signal. DRL scales RL to problems with high dimension state and action spaces by means of neural networks [10]. Most DRL algorithms consider problems under the Markov Decision Process (MDP) setting. Given a state space $\mathcal{S}$ and an action space $\mathcal{A}$, the MDP models interaction with an environment using a transition function $\mathcal{T}(s_t, a_t, s_{t+1}) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{P}$. The function $\mathcal{T}$ returns the probability of transitioning from a state s_t to a state s_{t+1} after taking an action a_t. Meanwhile, a function $R(s_t, a_t, s_{t+1})$ measures the transition reward. Given a discount factor $\gamma \in [0, 1]$, a controller $\pi : \mathcal{S} \to \mathcal{A}$ learns to maximize the discounted cumulative reward function $\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t, s_{t+1})$ where T is the total time horizon.

Many model-free on-policy ([11], [12]) and off-policy ([13], [14], [15]) DRL algorithms appeared in recent years. Among all these algorithms, our work is closely related to the DDPG [10]. An important component of DDPG is the Q function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. $Q(s_t, a_t)$ returns the discounted cumulative reward after taking action a_t under state s_t. A higher return of Q means a better action. Thus knowing $Q(s_t, a_t)$ allows us to choose the best action. In DDPG, a learned Q function serves as the critic for actions. An actor policy $\pi(s_t)$ is trained to maximize $Q(s_t, \pi(s_t))$. From the Q function we can derive the optimal policy being $\pi^*(s_t) = \arg\max_{\pi} Q(s_t, \pi(s_t))$.

B. Lyapunov Method

1) Lyapunov Function: A Lyapunov function characterizes the stability property of a controller. It is a function satisfying the following constraints:

$$V(s_o) = 0 \tag{1}$$
$$V(s_t) > 0, \forall s_t \neq s_o \tag{2}$$
$$V(s_{t+1}) - V(s_t) < 0 \tag{3}$$

In Eq. (1), a Lyapunov function's value is 0 at the origin s_o. s_o is a stabilized state of the controller. Eq. (2) enforces that the Lyapunov function is always positive when its input is not s_o. The L.H.S. of Eq. (3) is known as the lie derivative, $\nabla_\pi V = V(s_{t+1}) - V(s_t)$. When the lie derivative is smaller than 0, $V(s_t)$ strictly decreases over time. The lie derivative depends on controller π, since computing s_{t+1} requires action $a_t = \pi(s_t)$.

2) Neural Lyapunov Function: A neural Lyapunov function $V(s_t) : \mathcal{S} \to \mathbb{R}$ models a Lyapunov function via a neural network satisfying Eqs. (1)(2)(3). Training the NLF requires minimizing the Lyapunov risk [6], [7]. The Lyapunov risk is defined as

$$\begin{aligned} L_{lf}(\theta_V) = \mathbb{E}_{s_t \sim (E,\pi)}[&V_{\theta_V}^2(s_o) \\ &+ \max(0, -V_{\theta_V}(s_t)) \\ &+ \max(0, \nabla_\pi V_{\theta_V}(s_t))] \end{aligned} \tag{4}$$

Here, θ_V represents the parameters of the NLF with π being a controller. The Lyapunov risk is an expectation over s_t sampled from environment E and controller π. Minimizing $V_{\theta_V}^2(s_o)$ allow V_{θ_V} to satisfy Eq. (1), while minimizing $\max(0, -V_{\theta_V}(s_t))$ and $\max(0, \nabla_\pi V_{\theta_V}(s_t))$ makes V_{θ_V} satisfy Eq. (2) and Eq. (3) respectively. $\nabla_\pi V_{\theta_V}$ is the lie derivative over controller π as mentioned above.

3) Region of Attraction: The Lyapunov function specifies a *Region of Attraction* (RoA) as

$$\text{RoA} = \{s | V(s) < C_{RoA}\}, \tag{5}$$

where $C_{RoA} \in \mathbb{R}^+$ is a constant. Since the Lyapunov function strictly decreases over time, if we initialize a robot with any state in an RoA, the robot will always stay in the RoA in future. Since $V(s_o) = 0$, the origin s_o must be included in the RoA. It is also known as the sink of the RoA.

C. Path Planning

Sample-based planning methods like RRT [1] can be used to find a path to reach a goal state. RRT grows a collision-free tree from a given start state by randomly sampling the robot's configuration space. Once the tree finds a goal state, a Dijkstra algorithm extracts the shortest path connecting the given start and goal states for our navigation tasks.

D. Goal-Conditioned State Space

Goal-conditioned RL ([16]) augments the state space by conditioning it with a static goal $g \in \mathcal{G}$ where $\mathcal{G}$ is the goal space. In our problem, we consider a 2D goal space ($\mathcal{G} \subseteq \mathbb{R}^2$) representing a position in the 2D plane where a robot operates. Suppose the position of a robot under state s is $pos(s) \in \mathbb{R}^2$, we define a goal vector

$$d_g(s) = g - pos(s)$$

The state $s_g \in \mathcal{S}_g$ is a modified version of $s \in \mathcal{S}$ to contain this goal vector $d_g(s)$, and the intrinsic state $s_{/g}$ of a robot. The intrinsic state does not change as the robot position changes. Thus, every goal and state pair $(g, s) \in \mathcal{G} \times \mathcal{S}$ is mapped to

$$s_g(s) = [d_g(s), s_{/g}] \tag{6}$$

We train the controller and NLF in this goal-conditioned space to provide the flexibility of generalizing the learning-based components to different goals.

Since we must generalize the Lyapunov function to different goals as well, we always set the sink position $pos(s_o)$ to the current goal g. This effectively translates the Lyapunov function to be centered around an arbitrary goal g which means $pos(s_o) = g$ and the goal vector $d_g(s_o) = [0, 0]$ (when $\mathcal{G} \subseteq \mathbb{R}^2$). We choose this setting because our controller's objective is to arrive and be stable at the goal, while the Lyapunov function desires that the robot is asymptotically stabilized to s_o.

III. Approach

Fig. 2 depicts an overview of our approach. The navigation task requires guiding a robot to reach the final goal while avoiding collision with hazardous zones.

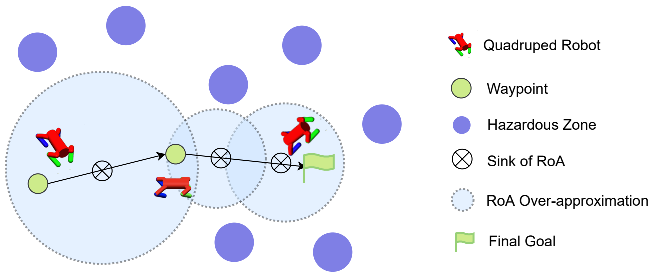


Fig. 2: Approach Overview: A robot is safely navigated from start to goal in over-approximated regions of attraction.

Our approach has three steps. First, we co-learn the controller and TNLF in a DRL loop. The controller learns to reach a given target in the shortest time. The design of the TNLF is introduced in Sec. III-A, and details about the co-learning procedure are given in Sec. III-B. Second, we run RRT on the 2D plane to build a collision-free path between the initial and goal positions. The path is a sequence of waypoints. Third, we use the learned controller to follow waypoints generated by the RRT planner and provide the safety guarantee with a runtime monitor. This runtime monitor places and builds the RoAs over-approximations with the learned TNLF, and it is introduced in Sec. III-C.

A. Twin Neural Lyapunov Function

The TNLF is a key component of our co-learning algorithm. Firstly, we define a function $\mathcal{Q}(s_t, a_t)$ to integrate the Lyapunov function into the DDPG training loop. $\mathcal{Q}(s_t, a_t)$ predicts the Lyapunov function value $V(s_{t+1})$ after taking action a_t. Then, similar to DDPG introduced in Sec. II-A, we can train our policy to minimize $\mathcal{Q}(s_t, \pi(s_t))$. Since $\mathcal{Q}$ works similarly to the Q function in DDPG, we name $\mathcal{Q}(s_t, a_t)$ the *Lyapunov Q function* and train $\mathcal{Q}(s_t, a_t)$ with a regression loss,

$$L_{lqf}(\theta_{\mathcal{Q}}) = ||\mathcal{Q}_{\theta_{\mathcal{Q}}}(s_t, a_t) - V_{\theta_V}(s_{t+1})||_2 \tag{7}$$

where s_{t+1} is the state that results from taking action a_t in s_t. Eq. (7) is only used to update $\theta_{\mathcal{Q}}$. The parameter θ_V is fixed when updating parameters with Eq. (7). Finally, we call $V(s_t)$ and $\mathcal{Q}(s_t, a_t)$ together as the twin neural Lyapunov function.

B. Co-learn the TNLF with controller

Co-learning the TNLF with the controller has significant benefits. First, when the Lyapunov function learns to characterize a controller, the controller is also adapted to the Lyapunov function. Thus, we can learn a better NLF to characterize stability properties. Second, the integration of the Lyapunov function can accelerate and stabilize controller convergence. This is because the Lyapunov function provides an additional training signal for RL. Similar ideas have also appeared in [14], [15], where the authors employ double Q functions for a better training signal.

Algorithm 1: Co-learn TNLF with Control Policy

```
Notions: Environment $E$, Policy $\pi$, Q function $Q$, Lyapunov Function $V$, Lyapunov Q function $\mathcal{Q}$, Training Network Parameter $\theta_{[\cdot]}$, Target Network Parameter $\theta'_{[\cdot]}$, Training Batch Size $N$, Polyak Constant $\tau$
1  Initialize training network $\pi_{\theta_\pi}, Q_{\theta_Q}, V_{\theta_V}, \mathcal{Q}_{\theta_{\mathcal{Q}}}$;
2  Create target network $\pi_{\theta'_\pi}, Q_{\theta'_Q}, \mathcal{Q}_{\theta'_{\mathcal{Q}}}$;
3  Initialize all target networks parameters $\theta'_{[\cdot]} \leftarrow \theta_{[\cdot]}$;
4  for $i = 1, \ldots, T_{ep}$ do
5      Replay Buffer $\mathcal{R} \leftarrow$ Transitions$(E, \pi_{\theta'_\pi})$;
6      for $j = 1, \ldots, T_{grad}$ do
7          Data Batch $\mathcal{D} \leftarrow$ Sample$(\mathcal{R}, N)$;
8          TrainQ$(Q_{\theta_Q} \mid \mathcal{D})$;
9          TrainTNLF$(V_{\theta_V}, \mathcal{Q}_{\theta_{\mathcal{Q}}} \mid \mathcal{D})$;
10         TrainControlPolicy$(\pi_{\theta_\pi} \mid Q_{\theta'_Q}, \mathcal{Q}_{\theta'_{\mathcal{Q}}}, \mathcal{D})$;
11         PolyakUpdate$(\theta'_{[\cdot]} \mid \theta_{[\cdot]}, \tau)$;
12     end
13 end
```

Algorithm 1 describes our co-learning framework. The policy network π_{θ_π}, the Q function network Q_{θ_Q}, and their target networks $\pi_{\theta'_\pi}$ and $Q_{\theta'_Q}$ come from the original DDPG algorithm. We integrate our Lyapunov function network V_{θ_V} and Lyapunov Q function network $\mathcal{Q}_{\theta_{\mathcal{Q}}}$ into the co-learning loop with the functions and variables highlighted in red. The Lyapunov function network V_{θ_V} will be used in our downstream planning algorithm. The Lyapunov Q function network $\mathcal{Q}(s_t, a_t)$ serves as another critic for action a_t. As a critic, the $\mathcal{Q}(s_t, a_t)$ affects the stability of the whole training process. Thus, we also created a target network $\mathcal{Q}_{\theta'_\mathcal{Q}}$ to avoid it changing dramatically.

Line 1 to 3 of Algorithm 1 initialize all the networks' parameters. Line 5 samples the transitions with target policy network $\pi_{\theta'}$ and store these transitions to replay buffer $\mathcal{R}$. The loop starting from line 5 updates the parameters of each network. Line 7 samples training data batch $\mathcal{D}$ from replay buffer $\mathcal{R}$. Line 8 updates θ_Q with sampled data $\mathcal{D}$ with the approach in [10]. Line 9 trains the $V_{\theta_V}, \mathcal{Q}_{\theta_\mathcal{Q}}$ with $\mathcal{D}$. The loss function for V_{θ_V} is the Lyapunov risk defined in Eq. (4), and $\mathcal{Q}_{\theta_\mathcal{Q}}$ is trained with loss in Eq. (7). Line 10 trains controller π_{θ_π} with $Q_{\theta'_Q}, \mathcal{Q}_{\theta'_\mathcal{Q}}, \mathcal{D}$. The controller π_{θ_π} is trained with the loss

$$L_{\pi_{\theta_\pi}} = \mathbb{E}_{s_t}\left[-Q_{\theta'_Q}(s_t, \pi_{\theta_\pi}(s_t)) + \alpha \mathcal{Q}_{\theta'_\mathcal{Q}}(s_t, \pi_{\theta_\pi}(s_t))\right] \quad (8)$$

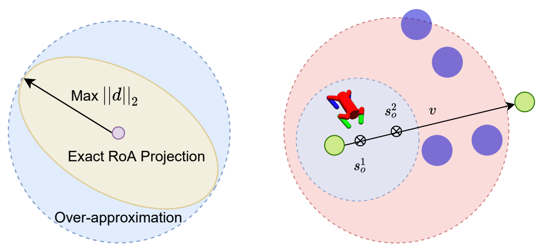
The training goal is to minimize $L_{\pi_{\theta_\pi}}$ via optimizing policy π_{θ_π}. Minimizing the $-Q_{\theta'_Q}$ term leads to maximizing the cumulative reward predicted by $Q_{\theta'_Q}$. When minimizing $\mathcal{Q}_{\theta'_\mathcal{Q}}$, the controller is learning to minimize the value of the Lyapunov function in the next step, which adapts the controller to satisfy Eq. 3. Here, α is a hyperparameter that controls the $\mathcal{Q}$ impact on the update of policy π. Line 11 conducts a Polyak update on the target networks for enhanced learning stability [10]. It updates all the target parameters with

$$\theta'_{[\cdot]} = (1-\tau)\theta'_{[\cdot]} + \tau\theta_{[\cdot]}$$

where $\tau \in (0, 1]$, and $[\cdot]$ can be $\pi, \mathcal{Q}$ or Q.

C. Runtime Monitor

Algorithm 1 provides us with a controller π and an NLF V characterizing π. We then run the RRT planner to generate a collision-free path to reach the final goal. A robot follows this path with the controller π, while its safety is ensured by a sequence of RoAs placed over the path. How to build these RoAs and where to place them are the two problems we address.



(a) RoA Over-approximation (b) Search Sink

Fig. 3: Our runtime monitor builds and aligns over-approximated RoAs around planner's waypoints.

a) Building RoA over-approximations: The runtime monitor builds the RoA over-approximation for the exact RoA projection on the 2D path plane. Given an NLF V, we can build an RoA in the state space $\mathcal{S}$. Because we only care about the collision in the 2D path plane, we project the RoA to the path plane. If this projection does not collide with a hazardous zone, the RoA inside must not collide with this hazardous zone. However, computing the exact projection can be hard. Instead, we propose an approach to compute the over-approximation of this projection with a demonstration provided in Fig. 3a. The over-approximation is a superset of the reachable set. Thus, the corresponding reachable set is guaranteed to avoid hazardous zones if the over-approximation does not intersect with them. Given a NLF V and a constant C_{RoA}, the RoA is specified as $\{s \mid V(s) < C_{RoA}\}$. The over-approximation we build is a circle around the projected RoA in the 2D path plane. If we place the center of this circle in the position of the sink (the current robot goal g), the radius we need to compute is the max L2-norm of the goal vector $d_g(s)$ for all the states in this RoA. We search the max $||d_g(s)||_2$ by maximizing the objective function in Eq. (9).

$$L_{ap}(s) = ||d_g(s)||_2 + \beta \min(C_{RoA} - V(s), 0) \quad (9)$$

When $\beta \in \mathbb{R}^+$ is a large positive constant, $\beta \min(C_{RoA} - V(s), 0)$ is a constraint which forces the search to stay in the RoA $\{s \mid V(s) < C_{RoA}\}$. When s is in the RoA specified by V and s_o, then $C_{RoA} \geq V(s)$ and $\min(C_{RoA} - V(s), 0)$ will be 0. Otherwise, $\min(C_{RoA} - V(s), 0)$ will be negative and penalizes the objective function. We optimize

$$s^* = \underset{s \in \mathcal{S}}{\arg\max}\ L_{ap}(s) \quad (10)$$

where we choose a large β to enforce $\min(C_{RoA} - V(s), 0) = 0$. We sample a batch of initial optimization states from $\mathcal{S}$ and optimize them with projected gradient descent [17] to estimate a solution to Eq. (10). Since $s^* = [d_g(s^*), s^*_{/g}]$, we can compute the radius as $||d_g(s^*)||_2$.

b) Placing RoA over-approximation: Fig. 3b shows how the runtime monitor places the sink and RoA over-approximations along the planned path. We prefer a larger RoA over-approximation because smaller regions usually result in more conservative behavior and make the robot move slower. However, if these RoA over-approximations intersect with the hazardous zones, the robot may violate safety properties. Therefore, our goal is to find the largest over-approximation that has no intersection with the hazardous zones. Given a vector $v = g_2 - g_1 \in \mathbb{R}^2$ from one waypoint $g_1 \in \mathcal{G}$ to the next waypoint $g_2 \in \mathcal{G}$, we search the positions of sinks between g_1 and g_2 using a line search. The position of a sink is given by

$$pos(s_o^i) = g_1 + i\delta v$$

where $i \in \mathbb{N}^+$, δ is a number controlling the granularity of the search, $\delta \in (0, 1]$. Given a state s_t, we build the smallest RoA, denoted as RoA*, which includes s_t.

$$\text{RoA}^*(s_t) = \{s \mid V(s) \leq V(s_t)\} \quad (11)$$

The state s_t depends on the position of sink $pos(s_o^i)$ (i.e. goal of a goal-conditioned state space). Hence, choosing different sink positions results in different RoA over-approximation as demonstrated in Fig. 3b. To find the largest over-approximation that has no intersection with hazardous zones, we need to compute the radius of over-approximation for every sink position via optimizing Eq. 9. Finally, we select the sink position with the largest radius and set it as the goal of the robot. We repeat this line searching in every step. Intuitively, this makes the RoA over-approximation change adaptively based on surrounding hazardous zones.

c) Pre-computed RoA-overapproximations: Because computing the over-approximation in real-time can be computationally expensive, in practice, we pre-compute and reference the radius of over-approximation via a lookup table. Eq.(5) tells us that the shape of the RoA only depends on C_{RoA} when given a trained NLF V. Thus, one C_{RoA} can only correspond to one minimal circle over-approximation. We pre-compute a lookup table that maps the C_{RoA} to its corresponding radius of the minimal circle over-approximation. According to Eq. (11), the C_{RoA} of $\text{RoA}^*(s_t)$ is determined by $V(s_t)$. However, because the lookup table is finite and cannot capture all possible C_{RoA}s, we may over-approximate an RoA with a larger circle instead of the exact one. In other words, we guarantee pre-computed over-approximations are supersets of corresponding RoAs. It is also noticeable that the pre-computation does not depend on the map. Thus, it is only a one-time effect and can generalize to any map.

Theorem 1: Suppose the lookup table has keys $C_{RoA}^1, \cdots C_{RoA}^N$ in increasing order, given a state s_t, and $C_{RoA}^i < V(s_t) \leq C_{RoA}^{i+1}$, the lookup table returns the radius corresponding to C_{RoA}^{i+1}. The over-approximation built with the returned radius must over-approximate $\text{RoA}^*(s_t)$.

Proof:

$$\text{RoA}^*(s_t) = \{s \mid V(s) \leq V(s_t)\}$$
$$\overline{\text{RoA}} = \{s \mid V(s) \leq C_{RoA}^{i+1}\}$$

$V(s_t) \leq C_{RoA}^{i+1} \implies \text{RoA}^*(s_t) \subseteq \overline{\text{RoA}}$. Suppose a point $p \in \text{RoA}^*(s_t)$ has the largest L2-norm to the sink s_o. $||p - pos(s_o)||_2$ is the over-approximation radius of $\text{RoA}^*(s_t)$. $\text{RoA}^*(s_t) \subseteq \overline{\text{RoA}} \implies p \in \overline{\text{RoA}}$. Thus, the over-approximation radius of $\overline{\text{RoA}}$ is at least $||p - pos(s_o)||_2$. Hence, the over-approximation of $\overline{\text{RoA}}$ must over-approximates $\text{RoA}^*(s_t)$. ∎

According to Theorem 1, the lookup table can always return the radius corresponding to C_{RoA}^{i+1} for building a circle over-approximation of $\text{RoA}^*(s_t)$. The pre-computed over-approximation can boost computational speed. Over-approximating an RoA will only require one forward computing on the neural network model V and a query in the lookup table. In this way, our algorithm can be effectively deployed online.

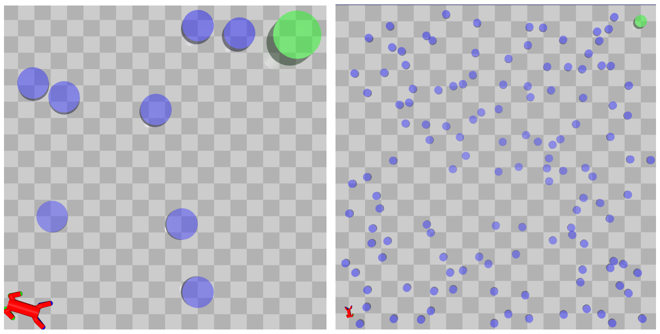
IV. Experiments

Our experiments aim to answer the following questions:

- Is co-learning a viable way to learn a performant policy while gaining a high-quality Lyapunov function?
- When compared with reward shaping and Lagrangian-based safe reinforcement learning approaches (CPO in our experiments), can we leverage the learned Lyapunov function for more safety-oriented tasks?
- Does our safety-oriented framework sacrifice the completion of the objective and the speed of reaching the goal in favor of safety?

A. Setup

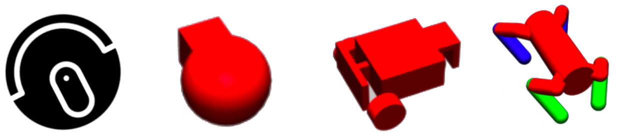
We show results on both a custom 2D sweeping robot environment as well as robotic environments in Safety Gym [9] with continuous state and action spaces.



(a) Level-1 Map (b) Level-3 Map

Fig. 4: Difficulty Levels of Navigation Tasks

The benchmarks for each robot were classified into three levels of difficulty based on the number of hazardous zones and map size as shown in Fig.1 and Fig.4. The initial and goal positions are placed on the map's lower-left corner and upper-right corner, respectively. The hazardous zone positions are initialized randomly for each run. The map size for difficulty level-1 to level-3 are $4 \times 4, 8 \times 8$ and 16×16, respectively. Level-1 has 8 hazardous zones, and level-2 and level-3 have 32 and 128 hazardous zones, respectively.



(a) *Sweeping* (b) *Point* (c) *Car* (d) *Quadruped*

Fig. 5: Our dynamical systems include *sweeping*, *point*, *car*, and *quadruped*. Their corresponding state and action space dimensions, denoted as (state dim, action dim), from (a)-(d) are (2,2), (14,2), (26,2) and (58,12), respectively.

Fig.5 depicts our dynamical models with their state and action space dimensions. *Sweeping* is a customized environment for moving the position of a sweeping robot in a 2-D plane. *Point* models a robot restricted to the 2-D plane with actuators to rotate and move forward. *Car* has two independently operated parallel wheels with a rear-wheel for balance. *Quadruped* models a quadrupedal robot with each leg having torque controls in the hip and knee joints. All agents obtain their state information from the joints, accelerometer, gyroscope, magnetometer, velocimeter, and a 2D vector toward the goal.

B. Co-learning

The low-level controller and NLF are trained in an environment without hazardous zones. Fig. 6 shows the learning progress for each of our robots in this hazardous zone-free environment. The reward used for training the controller is defined as

$$r_t = ||g - pos(s_t)||_2 - ||g - pos(s_{t+1})||_2 \tag{12}$$

where r_t is the reward at time t, g is the goal position the controller should achieve, $pos(s_t)$ and $pos(s_{t+1})$ are the positions of robot at time t and $t{+}1$, respectively. This reward is positive when $pos(s_{t+1})$ is closer to goal than $pos(s_t)$.

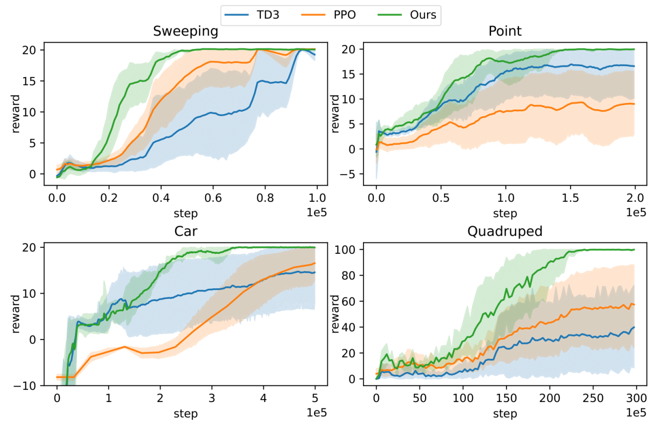


Fig. 6: Training performance depicting total accumulated rewards over the number of interaction steps.

Fig. 6 compares the training reward between our co-learning algorithm with the off-policy TD3 [14] and on-policy PPO [12]. Our co-learning algorithm can reach higher rewards in fewer simulation steps, while it is also more stable after the reward converges on all four robots. The experiments show that the training signal from the Lyapunov Q function critic can benefit the controller's training process.

TABLE I: Eq. (2) (3) Satisfaction Rate of NLF

Phase	*Sweeping*	*Point*	*Car*	*Quadruped*
Co-learn	**99.67 %**	**98.97 %**	**97.06 %**	**97.59 %**
Post-learn [7], [18]	99.42%	96.77 %	94.54%	93.47 %

The quality of NLF is measured with Eq. (1) (2) (3). For Eq. (1), all the sinks have $|V(s_o)| < 10^{-3}$ on the four robots. We generated 100,000 sampled state transitions (s_t, s_{t+1}) for each robot. These transitions are generated with the trained controller π. We evaluated these transitions with Eq. (2) (3). The results are reported in Table I. For any transition, (s_t, s_{t+1}), Eq. (2) requires that $V(s_t)$ and $V(s_{t+1})$ should be greater than 0. Eq. (3) requires that the lie derivative should be smaller than 0. A desired NLF should make all the transitions sampled from π satisfy Eq. (2) (3). We compared the NLF trained by our co-learning algorithm with the NLF trained in the post-learning phase [7], [18]. Because the co-learning algorithm can adapt the controller to the NLF during training, the NLFs trained by the co-learning algorithm have better quality with respect to satisfaction rate.

C. Baselines

Our approach explicitly avoids a robot entering a hazardous zone through the use of a runtime monitor. We compare two other existing model-free approaches that only implicitly encode avoidance behavior. We further evaluate these two algorithms with the RRT guidance for a fair comparison.

a) End-to-End (E2E): These RL controllers are trained with TD3 [14]. Here safety violations are encoded into the reward via a penalty term. The reward at time t is given by $r_t^{e2e} = r_t - C\mathbb{1}_{haz}$, where r_t is defined in Eq. (12), and $\mathbb{1}_{haz}$ is an indicator that shows whether a robot is in the hazardous zone. $C \in \mathbb{R}^+$ is a hyperparameter requiring tuning. The E2E is not guided by a planner and cannot avoid hazardous zones by following the planned path. Hence, we need to augment its observations with a vector $v_{haz} \in \mathbb{R}^{16}$ that provides position information of hazardous zones, allowing the controller to learn to avoid them. v_{haz} contains 8 vectors that point toward the 8 closest hazardous zones. The input of the E2E controller is $[v_{haz}, s_t]$.

b) Constrained Policy Optimization: Constrained Policy Optimization (CPO) [19] is a trust-region method for solving the constrained MDP problem. By constraining safety violations, [19] trains controllers in Safety Gym environments. However, the number of hazardous zones and map size described in the original CPO paper is limited compared with our approach. We observe that CPO can result in increasingly conservative behaviors as the number of hazardous zones and map sizes increase. CPO also needs to learn avoidance behaviors from information on hazardous zones directly. Thus, the input of the CPO controller is also $[v_{haz}, s_t]$.

c) Integrating Trained Controller with Planning: Both the E2E and CPO are trained in the goal-conditioned state space, and as a consequence, we can also guide them with a sequence of waypoints to reach the final goal. We use the same RRT algorithm to generate the waypoints to guide the controllers trained with E2E and CPO. All the algorithms combined with the RRT planner are named H-XYZ, where XYZ is the algorithm used to train the controller.

D. Safety, Reach Rate, and Performance

We evaluate the trained controllers for the four robots on three difficulty levels. The max simulation steps for level-1 is 1,000; level-2 and level-3 are limited to 4,000 and 16,000 steps before termination, resp. When a robot enters any hazardous zone, we terminate the simulation immediately and report a safety violation.

By comparing the fraction of episodes with safety violations on deployment (Table II), we observe a clear safety and reach rate benefit of our approach. In most cases, the safety violation ratios of all these algorithms are significantly

TABLE II: Safety Violation and Reach Rate

E2E : End-to-End, CPO : Constrained Policy Optimization [19], H: Hierarchical Planner
Safety violation and reach rate are fractions of 100 evaluation episodes with collisions and goal reachability, respectively.

Difficulty	Robot	Safety Violation ↓					Reach Rate ↑				
		E2E	CPO	H-E2E	H-CPO	H-Lyapunov	E2E	CPO	H-E2E	H-CPO	H-Lyapunov
level-1	*Sweeping*	0.13	0.05	0.27	0.32	**0.01**	0.87	0.75	0.73	0.68	**0.99**
	Point	0.66	0.05	0.08	0.21	**0.04**	0.25	0.07	0.72	0	**0.96**
	Car	0.27	0.19	0.41	0.18	**0.00**	0.73	0	0.59	0.22	**1**
	Quadruped	0.44	0.09	0.18	0.36	**0.05**	0.44	0	0.77	0.04	**0.95**
level-2	*Sweeping*	0.51	0.12	0.67	0.51	**0.01**	0.49	0.48	0.33	0.49	**0.99**
	Point	0.42	0.1	0.33	0.25	**0.05**	0.02	0	0.41	0	**0.95**
	Car	0.61	0.21	0.82	0.36	**0.00**	0.26	0	0.18	0.12	**1**
	Quadruped	0.04	0.06	0.35	0.53	**0.04**	0	0	0.41	0	**0.96**
level-3	*Sweeping*	0.88	0.23	0.83	0.84	**0.03**	0.12	0.17	0.15	0.14	**0.97**
	Point	0.09	0.12	0.45	0.33	**0.06**	0	0	0.09	0	**0.94**
	Car	0.79	0.28	0.98	0.38	**0.02**	0.02	0	0.02	0	**0.98**
	Quadruped	0.09	0.09	0.44	0.59	**0.08**	0	0	0.1	0	**0.92**

higher than our approach. However, there also exist cases with low safety violations on other algorithms. Nevertheless, one should note that a low violation rate is independent of the policy's performance and goal reach rate. A low safety violation rate may also mean a non-performant or inactive controller. For example, the *Quadruped* E2E controller under level-2 has a 0.04 safety violation ratio, while it also cannot reach the goal in all the 100 episodes simulations. The *point* H-E2E controller has a 0.08 safety violation ratio with 0.72 reach rate under level-1. However, as difficulty increases, its safety violation ratio dramatically increases, and its reach rate decreases as well. Our method is safer and has a significantly greater goal-reach rate than any of the baselines.

Although combining RRT with the controller can significantly benefit certain scenarios (e.g., for the E2E *point* controller under level-1, the reach rate boosts from 0.25 to 0.72), Table II also shows that this is not a generally exploitable principle. Guided by the waypoints generated by RRT, the robot may have to visit more places before reaching the goal. These additional explorations can expose the robot to more hazardous zones. For example, for all the CPO *Quadruped* controllers, the safety violation ratio increases significantly after combining them with RRT.

TABLE III: Average Number of Steps to Reach Goal

'-' indicates complete failure in task achievement.

Difficulty	Robot	Steps to Reach				
		E2E	CPO	H-E2E	H-CPO	H-Lyapunov
level-1	*Sweeping*	120.7	121.4	201.7	192.1	234.7
	Point	403.6	793.8	409.6	-	521.9
	Car	381.1	-	444.2	765.5	417.0
	Quadruped	99.4	-	117.7	478.2	252.0
level-2	*Sweeping*	325.8	298.7	505.9	503.7	584.4
	Point	528.0	-	887.1	-	1015.2
	Car	720.7	-	858.7	2297.6	812.1
	Quadruped	-	-	242.1	-	500.5
level-3	*Sweeping*	764.0	675.4	1080.9	1097.9	1226.9
	Point	-	-	1498.4	-	1948.7
	Car	1212.5	-	1678.0	-	1573.8
	Quadruped	-	-	504.9	-	880.5

We evaluate the performance of algorithms with the steps-to-goal in Table III. While our method yields an enhanced safety and goal reach rate, we pay the price of having a larger mean of the number of steps to reach in most environments. This is expected as a more reckless controller may reach the goal faster, but this would be at the cost of safety. Our algorithm has larger steps to reach because of the small RoA over-approximations when a robot passes narrow tunnels. These small over-approximations cause cautious behaviors and make the robot move slower than usual. We also find that after combining with RRT, the robot needs more steps to reach the goal when compared with its non-hierarchical counterpart. This is normal because the paths generated with RRT are not guaranteed to be the shortest. This problem can be alleviated with algorithms like RRT^* [1]. However, since this paper mainly focuses on safety when a robot follows a planned path, the choice of planning algorithms is left for future work.

V. Related Work

Planning under kinodynamic constraints [2] addresses the inconsistency problem between planning and control. However, most previous works ([20], [21], [22], [23]) suffer from issues in scalability and generalization. Notably, compared with our work, [20] has a similar idea that combines stability region and planning. However, this approach only works for linearized known dynamics, and computing the LQR-tree is computationally expensive for deployment. More recently, [24] introduced a learning-based path planner and follows the planned path with MPC solved by Cross-Entropy Method (CEM) [25]. However, providing safety assurance is still challenging because the CEM avoids potential collisions via implicitly optimizing penalty terms encoded.

Integrating path planning with a DRL controller has been extensively explored recently ([3], [4], [5], [26]). For instance, [3] and [4] train a local point-to-point controller with DRL and employ probabilistic roadmaps and rapidly-exploring random trees as the high-level planner, respectively. [5] applies a convolutional neural network planner and trains the reach-avoidance controller with DRL. These approaches can work with unknown dynamics and high-dimensional raw sensor observations. However, the desired behaviors of these controllers are only specified in the reward functions and also inherit the limitations of standard RL.

Safe reinforcement learning approaches ([19], [27], [28], [29], [30], [31]) aim at learning controllers that causes limited constraint violations. [19], [27], [28] can reduce violation numbers but also results in more conservative behaviors, which can affect performance. [29], [30], [31] can avoid all violations. However, they require a safe controller and the dynamics to be known, which is a strong requirement compared to our model-free setting. Moreover, since these approaches are not integrated with any planner, they usually do not perform well for long-time-horizon navigation tasks.

The Lyapunov method is also applied in [28], [29], [30], [31]. However, [28] does not build an explicit Lyapunov function. [30], [31] compute the Lyapunov function with analytic approaches instead of learning it. Hence, they suffer from scalability issues and require knowing the controller's dynamics. In contrast, [7], [8], [32], [6] model the Lyapunov function with a neural network, training it with the transitions sampled from the environment and controller. [7], [32], [6] build the NLF with a fixed controller, and thus unlike our work, cannot be used to improve the quality of the controller. [8] also co-learns the Lyapunov function and controller, but with supervised learning, requiring a training dataset in advance.

VI. Conclusion

In this paper, we first introduced the TNLF and subsequently, using information derived by modeling this function for a DRL controller, demonstrate a clear enhancement to safety aspects when following plans. Additionally, we prove that learning these controllers along with the TNLF can be done via a novel co-learning procedure yielding benefits to both components. We show that general planning algorithms, albeit capable, are in themselves inherently lacking in their ability to avoid safety violations, and the execution strategies for these plans must also take into account the capabilities of the underlying controller. Our use of RoAs and runtime monitors is a significant leap towards realizing this synergy between safety and planning.

References

[1] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[2] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM (JACM)*, vol. 40, no. 5, pp. 1048–1066, 1993.

[3] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson, "Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018.

[4] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4298–4305, 2019.

[5] K. Ota, Y. Sasaki, D. K. Jha, Y. Yoshiyasu, and A. Kanezaki, "Efficient exploration in constrained environments with goal-oriented reference path," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 6061–6068.

[6] S. M. Richards, F. Berkenkamp, and A. Krause, "The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems," in *Conference on Robot Learning*, 2018.

[7] Y.-C. Chang, N. Roohi, and S. Gao, "Neural lyapunov control," *arXiv preprint arXiv:2005.00611*, 2020.

[8] D. Sun, S. Jha, and C. Fan, "Learning certified control using contraction metric," *arXiv preprint arXiv:2011.12569*, 2020.

[9] A. Ray, J. Achiam, and D. Amodei, "Benchmarking safe exploration in deep reinforcement learning," 2019.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning." in *ICLR (Poster)*, 2016.

[11] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *ICML*, 2015.

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

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[14] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*, 2018.

[15] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, 2018.

[16] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[17] Y. Chen and M. J. Wainwright, "Fast low-rank estimation by projected gradient descent: General statistical and algorithmic guarantees," *arXiv preprint arXiv:1509.03025*, 2015.

[18] Z. Xiong, I. Agarwal, and S. Jagannathan, "Hisarl: A hierarchical framework for safe reinforcement learning," *AAAI SafeAI Workshop*, 2022.

[19] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *ICML*, vol. 70, 06–11 Aug 2017.

[20] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "Lqr-trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.

[21] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 528–564, 2016.

[22] B. Sakcak, L. Bascetta, G. Ferretti, and M. Prandini, "Sampling-based optimal kinodynamic planning with motion primitives," *Autonomous Robots*, vol. 43, no. 7, pp. 1715–1732, 2019.

[23] C. Xie, J. van den Berg, S. Patil, and P. Abbeel, "Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[24] L. Li, Y. Miao, A. H. Qureshi, and M. C. Yip, "Mpc-mpnet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4496–4503, 2021.

[25] Z. I. Botev, D. P. Kroese, R. Y. Rubinstein, and P. L'Ecuyer, "The cross-entropy method for optimization," in *Handbook of statistics*. Elsevier, 2013, vol. 31, pp. 35–59.

[26] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2007–2014, 2019.

[27] Y. Chow, M. Ghavamzadeh, L. Janson, and M. Pavone, "Risk-constrained reinforcement learning with percentile risk criteria," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6070–6120, 2017.

[28] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh, "A lyapunov-based approach to safe reinforcement learning," *NeurIPS*, vol. 31, 2018.

[29] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," *Advances in neural information processing systems*, vol. 30, 2017.

[30] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *AAAI*, 2019.

[31] H. Zhu, Z. Xiong, S. Magill, and S. Jagannathan, "An inductive synthesis framework for verifiable reinforcement learning," in *PLDI*, 2019.

[32] V. Petridis and S. Petridis, "Construction of neural network based lyapunov functions," in *IJCNN*. IEEE, 2006.