NavRL: Learning Safe Flight in Dynamic Environments

Zhefan Xu, *Graduate Student Member, IEEE*, Xinming Han, *Graduate Student Member, IEEE*, Haoyu Shen, Hanyu Jin, and Kenji Shimada

***Abstract*—Safe flight in dynamic environments requires unmanned aerial vehicles (UAVs) to make effective decisions when navigating cluttered spaces with moving obstacles. Traditional approaches often decompose decision-making into hierarchical modules for prediction and planning. Although these handcrafted systems can perform well in specific settings, they might fail if environmental conditions change and often require careful parameter tuning. Additionally, their solutions could be suboptimal due to the use of inaccurate mathematical model assumptions and simplifications aimed at achieving computational efficiency. To overcome these limitations, this letter introduces the NavRL framework, a deep reinforcement learning-based navigation method built on the Proximal Policy Optimization (PPO) algorithm. NavRL utilizes our carefully designed state and action representations, allowing the learned policy to make safe decisions in the presence of both static and dynamic obstacles, with zero-shot transfer from simulation to real-world flight. Furthermore, the proposed method adopts a simple but effective safety shield for the trained policy, inspired by the concept of velocity obstacles, to mitigate potential failures associated with the black-box nature of neural networks. To accelerate the convergence, we implement the training pipeline using NVIDIA Isaac Sim, enabling parallel training with thousands of quadcopters. Simulation and physical experiments show that our method ensures safe navigation in dynamic environments and results in the fewest collisions compared to benchmarks.**

***Index Terms*—Aerial systems: Perception and autonomy, reinforcement learning, collision avoidance.**

I. Introduction

AUTONOMOUS unmanned aerial vehicles (UAVs) are widely used in applications like exploration [1], search and rescue [2], and inspection [3]. These tasks often occur in dynamic environments that require effective collision avoidance. Traditional methods rely on handcrafted algorithms and hierarchical modules, leading to overly complex systems with hard-to-tune parameters. In contrast, reinforcement learning (RL) allows UAVs to learn decision-making through experience, offering better adaptability and improved performance.

Received 23 September 2024; accepted 11 February 2025. Date of publication 26 February 2025; date of current version 10 March 2025. This article was recommended for publication by Associate Editor G. Pizzuto and Editor A. Faust upon evaluation of the reviewers' comments. *(Corresponding author: Zhefan Xu.)*

The authors are with the Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: zhefanx@andrew.cmu.edu).

Experiment video link: https://youtu.be/EbeJW8-YlvI

This article has supplementary downloadable material available at https://doi.org/10.1109/LRA.2025.3546069, provided by the authors.

Digital Object Identifier 10.1109/LRA.2025.3546069

Fig. 1. A customized quadcopter UAV navigating a dynamic environment using the proposed NavRL framework. The robot achieves safe navigation and effective collision avoidance with both static and dynamic obstacles.

Developing RL-based navigation methods is essential for enhancing UAV safety in dynamic environments.

Developing an RL-based navigation method suitable for real-world deployment presents several challenges. First, since reinforcement learning involves training the robot through collision experiences, the learning process must occur in simulated environments. This creates a sim-to-real transfer issue due to the gap between simulated and real-world sensory information, particularly with camera images. Previous works have attempted to address this issue by developing methods to reduce the gap between simulated and real-world environments [4], [5], [6] or by training the robot directly in real-world settings [7], [8], [9]. However, these approaches often require additional training steps and can be data-inefficient when trained in real-world scenarios. Second, even if the trained RL policy demonstrates satisfactory performance, ensuring safety remains challenging due to the black-box nature of neural networks, necessitating an effective safety mechanism to prevent severe failures [10], [11], [12], [13]. Lastly, training a reinforcement learning policy requires a substantial amount of robot experience, and some previous methods [14], [15], [16] that collect data using a single robot often result in slow convergence speeds due to limited data diversity and reduced parallel exploration opportunities.

To address these issues, this letter proposes a deep reinforcement learning-based navigation framework, named NavRL, based on the Proximal Policy Optimization (PPO) algorithm [17]. The proposed framework employs state and action representations built on our perception module, specifically designed for collision avoidance in dynamic environments, enabling zero-shot sim-to-real transfer capability. Additionally, to prevent severe failures, we leverage the concept of velocity

obstacles (VO) [18] to create a simple but effective safety shield using linear programming to optimize the RL policy network's action outputs. To accelerate training convergence, we design a parallel training pipeline capable of simulating thousands of quadcopters simultaneously using NVIDIA Isaac Sim. We validate the proposed framework through extensive simulation and physical flight experiments in various environments, demonstrating its ability to ensure safe navigation. Fig. 1 illustrates an example of our UAV navigating in a dynamic environment using the proposed NavRL framework. The main contributions of this work are:

- *The NavRL Navigation Framework:* This work introduces a novel reinforcement learning-based UAV navigation system to ensure safe autonomous flight in dynamic environments. The NavRL navigation framework is made available as an open-source package on GitHub.[1]
- *Policy Action Safety Shield:* Our method adopts a safety shield into the policy network's action outputs to enhance safety based on the velocity obstacle concept.
- *Physical Flight Experiments:* Real-world experiments in various environments are conducted to demonstrate the safe navigation capabilities and zero-shot sim-to-real transfer effectiveness of the proposed method.

II. Related Work

Research on UAV navigation in dynamic environments often relies on rule-based methods with handcrafted algorithms [19], [20], [21], which can be complex and require careful tuning as conditions change. In contrast, learning-based methods reduce complexity and adapt better to varying environments. This section mainly categorizes learning-based navigation into supervised and reinforcement learning-based methods, while acknowledging the existence of other approaches.

Supervised learning-based methods: Methods in this category train networks using labeled datasets. Early approaches [8], [22] deploy robots in real-world environments, collect images, and manually label them with ground truth actions. Similarly, some methods [7], [23] predict the safety of input images rather than outputting the decision and then use handcrafted algorithms to control the robot.

The methods mentioned above require manually labeled real-world data and often suffer from limited generalization ability due to insufficient data. Loquercio et al. [24] train the network on an autonomous driving dataset to achieve navigation, benefiting from its extensive data volume. Jung et al. [25] use a learning-based detector to enable a drone to pass through gates in racing, while in [26], iterative learning control is applied to reduce tracking error in this context. Works on foundation models for visual navigation has been developed using real-world experience from various robots [27], [28], [29]. Simon et al. [30] demonstrate collision avoidance capabilities for drones equipped with monocular cameras using a depth estimation method [31]. An imperative learning approach based on semantic images is proposed in [32] to achieve semantically-aware local navigation.

[1]Software available at: https://github.com/Zhefan-Xu/NavRL

Reinforcement learning-based methods: Compared to supervised learning-based methods, reinforcement learning-based approaches benefit from the abundant data in simulation. Some methods utilizing Q-learning [14], [15], [33] or value learning [34], [35] have demonstrated successful navigation. However, these methods are constrained to discrete action spaces, which can lead to suboptimal performance.

Policy gradient methods [16], [36], using an actor-critic structure, have gained popularity for enabling robot control in continuous action spaces. Kaufmann et al. [37] trained a policy in simulation to outperform human champions in drone racing, with other studies enhancing control performance [38], [39]. Song et al. [6] developed a vision-based network by distilling knowledge from a teacher policy with privileged information, while Xing et al. [40] applied contrastive learning to improve image encoding. In [41], a comprehensive reinforcement training pipeline is presented for UAV-related tasks.

To ensure safety, a recovery policy using a reach-avoid network is introduced in [42] to prevent failures. Similarly, Kochdumper et al. [13] utilize reachability analysis to project unsafe policy actions to safe regions. However, their approach requires a precomputed reachability set, with computation costs scaling exponentially with action dimensions.

Most existing methods are designed for navigation and collision avoidance in static environments. Some methods, such as [36], demonstrate dynamic obstacle avoidance but do not address collision avoidance in complex static scenarios. The method in [42] can handle both static and dynamic obstacles but may perform suboptimally by treating dynamic obstacles as static. Recent safety-focused approaches are either computationally expensive or require extensive training. These challenges, combined with the difficulty of sim-to-real transfer, motivate us to propose a framework that ensures safe navigation while avoiding both static and dynamic obstacles.

III. Methodology

The proposed NavRL navigation framework is depicted in Fig. 2. The obstacle perception system processes RGB-D images and robot states to generate representations for both static and dynamic obstacles (Section III-A). Section III-B details the conversion of these obstacle representations into network input states, along with the definitions of robot actions and training rewards. During training, we employ the PPO algorithm with an actor-critic network structure to train the policy and value networks, as explained in Section III-C. For deployment, a safety shield is applied to the RL policy network's action outputs to ensure safety (Section III-D).

A. Obstacle Perception System

Static and dynamic obstacles are handled separately by our perception system due to their different properties, as illustrated in Fig. 2. Static obstacles, which can have arbitrary shapes and sizes, are represented more accurately using a discretized format, such as an occupancy voxel map. Conversely, dynamic obstacles,

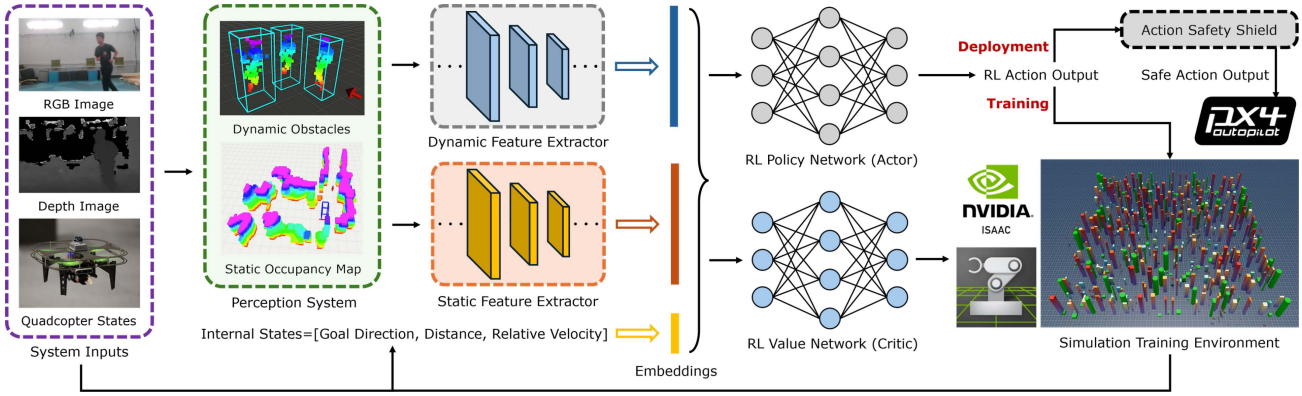


Fig. 2. The proposed NavRL framework. The perception system processes RGB-D images along with the robot's internal states to generate representations for both static and dynamic obstacles. These representations are then fed into two feature extractors, which produce state embeddings concatenated with the robot's internal states. In the training phase, an actor-critic network structure is utilized to train robots in parallel within the NVIDIA Isaac Sim environment. During the deployment stage, the policy network generates actions that are further refined by a safety shield mechanism to ensure safe robot control.

typically modeled as rigid bodies, are represented using bounding boxes with estimated velocity information. The rationale for using different representations for static and dynamic obstacles lies in their distinct characteristics. Static obstacles sometimes have irregular geometries, making a discrete representation more suitable for accurate shape description. In contrast, dynamic obstacles, typically rigid bodies, require a representation that captures their states as a whole entity. Additionally, updating the occupancy map with dynamic obstacles can introduce noise and latency, further justifying the need for separate representations. For the perception of static obstacles, we create a 3D occupancy voxel map with a fixed memory size, determined by the maximum number of voxels suitable for the environment. The occupancy data for each voxel is then stored in a pre-allocated array. This design enables us to access occupancy information with constant time complexity ($\mathcal{O}(1)$). At each time step, we recursively update the log probability of occupancy for each voxel based on the latest depth image and clear the occupancy data for dynamic obstacles by iterating through their detected bounding boxes. It is important to note that this static occupancy map can be generated on the fly without the need for any prebuilt data.

Detecting 3D dynamic obstacles is challenging due to the noisy depth images from the lightweight UAV camera and the limited processing power of the onboard computer. To achieve accurate detection with minimal computational demand, we propose an ensemble method that combines two lightweight detectors built on [43]. The first, the U-depth detector [44], converts raw depth images into a U-depth map (similar to a top-down view) and uses a contiguous line grouping algorithm to detect 3D bounding boxes of obstacles. The second, named the DBSCAN detector, applies the DBSCAN clustering algorithm to point cloud data from depth images to identify obstacle centers and dimensions by analyzing boundary points within each cluster. Both detectors, however, can produce a significant number of false positives due to the noisy input data. Our proposed ensemble method addresses this by finding the mutual agreements from both detectors to identify consistent results. Furthermore, to differentiate between static and dynamic obstacles, we employ a lightweight YOLO detector to classify dynamic obstacles by examining the 2D bounding boxes that are re-projected onto the image plane from the 3D detections.

The dynamic obstacle velocity is estimated by the tracking module, which operates in two stages: data association and state estimation. In the data association stage, the goal is to establish correspondences between detected obstacles across consecutive time steps. To minimize detection mismatches, we construct a feature vector for each obstacle that includes its position, bounding box dimensions, point cloud size, and point cloud standard deviation, and then determine matches based on the similarity scores between potential matching obstacles. In the state estimation stage, a Kalman filter is used to estimate obstacle velocities, with a constant acceleration model applied to account for changes in velocity over time. It is worth noting that while the perception system in this framework is based on a camera, other sensors can also be used if they provide a similar obstacle representation. The dynamic obstacle detection and tracking in the perception system is based on the approach presented in [43], where readers can find further details and implementation specifics.

B. Reinforcement Learning Formulation

The navigation task is formulated as a Markov Decision Process (MDP) defined by the tuple (S, A, P, R, γ), where S is the state space (robot's internal and sensory data), A is the action space, the transition function $P(s_{t+1}|s_t, a_t)$ models environment dynamics, and the reward function $R(s_t, a_t)$ encourages goal-reaching behaviors while penalizing collisions and inefficient actions. The goal is to learn an optimal policy $\pi^*(a_t|s_t)$ that maximizes the expected cumulative reward:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right], \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor for future rewards. This formulation enables the use of reinforcement learning to train the robot for safe navigation in dynamic environments.

State: As the input to the policy, the state must include all information relevant to navigation and collision avoidance. In our system framework shown in Fig. 2, the designed state is composed of three parts: the robot's internal states, dynamic obstacles, and static obstacles. The robot's internal states provide details about the robot's direction and distance to the navigation goal with its current velocity defined as:

$$S_{int} = \left[\frac{P_g^G - P_r^G}{\|P_g^G - P_r^G\|}, \|P_g^G - P_r^G\|, V_r^G \right]^T, \quad (2)$$

where P_r and P_g represent the robot position and goal position, respectively, and V_r is the robot current velocity. The superscript $(\cdot)^G$ indicates that the vector is expressed in the "goal coordinate frame", which is defined with its origin at the robot's starting position. In this frame, the x-axis aligns with the vector pointing from the starting position, P_s, to the goal position, while the y-axis lies parallel to the ground plane. This goal coordinate transformation reduces dependency on the global coordinate system, improving overall RL training convergence speed, and will also be applied to the definition of obstacle state representations.

The dynamic obstacles are represented using a 2D matrix:

$$S_{dyn} = [\mathcal{D}_1, \ldots, \mathcal{D}_{N_d}]^T, \; S_{dyn} \in \mathbb{R}^{N_d \times M}, \; \mathcal{D}_i \in \mathbb{R}^M. \quad (3)$$

In this formulation, $\mathcal{D}_i$ denotes the state vector of the ith closest dynamic obstacle to the robot and is expressed as:

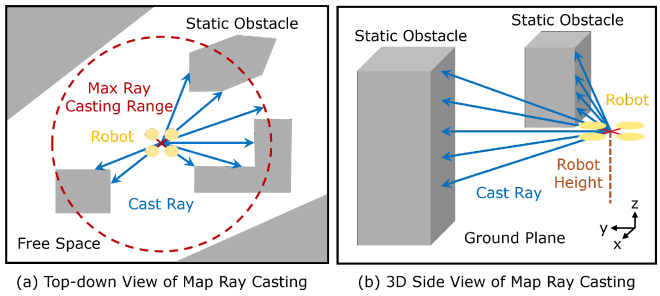
$$\mathcal{D}_i = \left[\frac{P_{o_i}^G - P_r^G}{\|P_{o_i}^G - P_r^G\|}, \|P_{o_i}^G - P_r^G\|, V_{o_i}^G, \dim(o_i) \right]^T, \quad (4)$$

where P_{o_i} and V_{o_i} represent the center position and velocity of the dynamic obstacle, respectively, and $\dim(o_i)$ indicates the height and width of the obstacle. The number of dynamic obstacles, N_d, is predefined, and if the actual number of detected obstacles is less than this predefined limit, the state vector values are set to zero. The relative position vectors for both internal states and obstacle states are split into a unit vector and its norm, as this method showed slightly faster convergence speed in our experiments.

Compared to dynamic obstacles, static obstacles are represented as map voxels, which cannot be directly input into neural networks. Therefore, we perform 3D ray casting from the robot's position against the map. In Fig. 3(a), rays are cast horizontally in all directions within a maximum range using a user-defined ray casting angle interval. Similarly, Fig. 3(b) illustrates the same operation in the vertical plane. For each diagonal ray casting angle θ_i in the vertical plane, the lengths of all rays in the horizontal plane are recorded into a vector R_{θ_i}. Any ray exceeding the maximum range is assigned a length equal to the maximum range plus a small offset, allowing obstacle absence to be identified. The representation of static obstacles is constructed by stacking the ray length vectors for all diagonal ray angles in the vertical planes:

$$S_{stat} = [R_{\theta_0}, \ldots, R_{\theta_{N_v}}], S_{stat} \in \mathbb{R}^{N_h \times N_v}, R_{\theta_i} \in \mathbb{R}^{N_h}, \quad (5)$$

where N_v and N_h represent the number of rays in the vertical and horizontal planes, determined by the ray casting angle interval and the vertical field of view. Unlike using image data as state representation, the proposed RL state representation has minimal discrepancies between simulation and the real world, which is beneficial for sim-to-real transfer.



(a) Top-down View of Map Ray Casting (b) 3D Side View of Map Ray Casting

Fig. 3. Illustration of map ray casting. Only rays within the maximum range are shown. (a) A top-down view of horizontally cast rays with a 360-degree casting angle. (b) A side view displaying rays in the vertical planes.

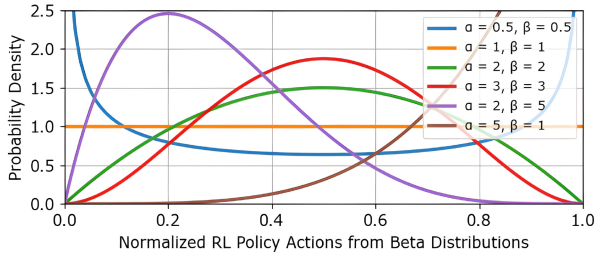


Fig. 4. Visualization of example RL policy actions from Beta distributions.

Action: At each time step, the velocity control $V_{ctrl} \in \mathbb{R}^3$ is provided to the robot for navigation and collision avoidance. Velocity control is chosen because higher-level controls offer better transferability and generalization across different platforms, facilitating sim-to-real transfer. Additionally, velocity commands are more interpretable and easier for humans to supervise compared to lower-level controls. Instead of directly outputting velocity values from the RL policy, the policy is designed to first infer a normalized velocity $\hat{V}_{ctrl}^G$ with the final output velocity expressed as:

$$V_{ctrl}^G = v_{lim} \cdot (2 \cdot \hat{V}_{ctrl}^G - 1), \; \hat{V}_{ctrl}^G \in [0, 1], \quad (6)$$

where v_{lim} is the user-defined maximum velocity. The final velocity is expressed in the goal coordinate frame, as described in the state formulation, requiring a coordinate transformation to be applied to the robot. This approach offers greater flexibility compared to other formulations where the RL policy must learn the action limits and cannot easily adjust the trained action range. To constrain the network output to the range $[0, 1]$, the model is designed to produce parameters (α, β) for a Beta distribution, as shown in Fig. 4. When the RL action space is constrained, a Beta distribution-based policy has been proven to be bias-free and achieve faster convergence compared to a Gaussian distribution-based policy [45]. During the training process, exploration is encouraged by sampling from the Beta distribution using the generated parameters. In deployment, we

use the mean of the Beta distribution as the normalized velocity output.

Reward: The designed RL reward function is computed at each time step consisting of multiple components:

$$r = \lambda_1 r_{vel} + \lambda_2 r_{ss} + \lambda_3 r_{ds} + \lambda_4 r_{smooth} + \lambda_5 r_{height}, \quad (7)$$

where $r_{(\cdot)}$ represents one type of reward weighted by λ_i. Each reward will be explained in the following paragraphs.

a) Velocity reward r_{vel}: The velocity reward encourages the robot to adopt velocities leading to the goal position:

$$r_{vel} = \frac{P_g - P_r}{\|P_g - P_r\|} \cdot V_r, \text{ where } P_g, P_s, V_r \in \mathbb{R}^3. \quad (8)$$

This formulation rewards velocities that align more closely with the position-to-goal direction with higher speeds.

b) Static safety reward r_{ss}: The static safety reward ensures that the robot keeps a safe distance from static obstacles. Given the static obstacle states in (5), it is defined as:

$$r_{ss} = \frac{1}{N_h N_v} \sum_{i=1}^{N_h} \sum_{j=1}^{N_v} \log S_{stat}(i,j). \quad (9)$$

This formulation computes the average log distance to static obstacles using the ray distances. The reward is maximized when the robot maintains greater distances from obstacles.

c) Dynamic safety reward r_{ds}: Similar to the static safety reward, the dynamic safety reward encourages the robot to avoid dynamic obstacles and is expressed as:

$$r_{ds} = \frac{1}{N_d} \sum_{i=1}^{N_d} \log \|P_r - P_{o_i}\|. \quad (10)$$

d) Smoothness reward r_{smooth}: The smoothness reward penalizes sudden changes in the control output, written as:

$$r_{smooth} = -\|V_r(t_i) - V_r(t_{i-1})\|, \quad (11)$$

where the L2 norm of the difference between the robot's velocities at the current and previous time steps is computed.

e) Height reward r_{height}: The height reward is designed to prevent the robot from avoiding obstacles by flying excessively high. It can be written as the following:

$$r_{height} = -(\min(|P_{r,z} - P_{s,z}|, |P_{r,z} - P_{g,z}|))^2, \quad (12)$$

which applies when the current height $P_{r,z}$ falls outside the range defined by the start height $P_{s,z}$ and goal height $P_{g,z}$.

C. Network Design and Policy Training

Given that our state representation consists of multiple components, a preprocessing step is necessary before inputting the data into the RL policy network. Both static and dynamic obstacles are represented as 2D matrices, so we utilize convolutional neural networks (CNN) to extract their features and transform them into 1D feature embeddings, as illustrated in Fig. 2. These embeddings are then concatenated with the robot's internal states to form the complete input feature for the policy and value networks. We employ the Proximal Policy Optimization (PPO) algorithm [17] to train the actor (policy) and critic (value) networks, both of which are implemented using multi-layer perceptron. The training process is conducted in NVIDIA Isaac Sim, where we implement parallel training by collecting data from thousands of quadcopters simultaneously. The training environment features a forest-like setting with both static and dynamic obstacles. Each robot is spawned at a random location with a random goal and is reset either upon colliding with an obstacle or at the end of the episode. To improve learning efficiency, we adopt a curriculum learning strategy. The environment initially starts with a relatively low obstacle density, which is gradually increased as the success rate surpasses a specified threshold. Our experiments show that this approach allows the RL policy to achieve a higher navigation success rate in complicated environments.

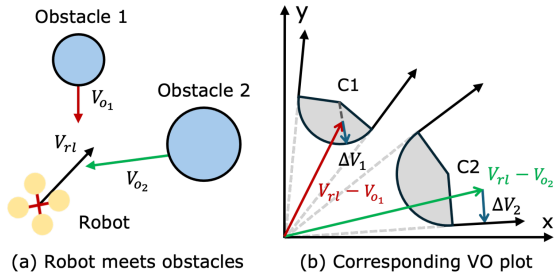


Fig. 5. Illustration of determining the safe velocity region using the velocity obstacle-based method. (a) An example scenario where the robot encounters two obstacles. (b) The corresponding velocity obstacle plot with blue arrows showing the minimum velocity change required to exit the VO regions.

D. Policy Action Safety Shield

Due to the black-box nature of the neural network, we designed a safety shield mechanism to prevent severe failures caused by the trained policy. Given that our policy outputs velocity commands for robot control, we incorporate the concept of velocity obstacles (VO) [18] to evaluate the safety of the policy's actions and correct them if they could result in collisions. The velocity obstacle represents the set of robot velocities that would result in a collision within a specified future time horizon. In the scenario depicted in Fig. 5(a), where the robot encounters obstacles, the corresponding velocity obstacle regions are visualized in Fig. 5(b). In this example, there are two obstacles moving at different velocities, each associated with a velocity obstacle region based on their relative positions and velocities with respect to the robot. Each velocity obstacle region combines a circular area centered at C1 or C2, with a radius equal to the sum of the robot and obstacle sizes plus user-defined safe space, and a cone area extending from the origin, excluding the dotted lines. The relative velocities of the robot to Obstacle 1 and Obstacle 2 (shown as red and green arrows) lie within the velocity obstacle regions, indicating future collisions. Inspired by [46], we adopt a similar approach to compute the minimum changes in velocity (ΔV_1 and ΔV_2) required to exit velocity obstacle regions. The policy action produced by the network is defined as the velocity V_{rl}. If this action is not within any of the velocity obstacle regions, we set the safe action V_{safe}

equal to the policy action. Otherwise, we formulate the following optimization to project the policy action into the safe region:

$$\min_{V_{safe}\in\mathbb{R}^3} \quad \|V_{safe} - V_{rl}\|, \tag{13a}$$

$$\text{s.t.} \quad (\mathrm{V_{safe}} - (\mathrm{V_{rl}} - \mathrm{V_{o_i}} + \Delta \mathrm{V_i})) \cdot \Delta \mathrm{V_i} \geq 0 \tag{13b}$$

$$V_{\min} \leq V_{safe} \leq V_{\max} \tag{13c}$$

$$\forall i \in \{1, \ldots, N\}, \tag{13d}$$

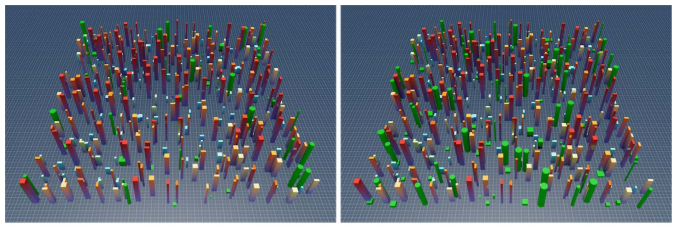
where (13b) defines a hyperplane based on the required changes in velocity to ensure the safe action lies outside the velocity obstacle regions, and (13c) enforces the control limits on the safe action. The drawback of constraining the safe action to one side of the hyperplane can be overly conservative when many obstacles are present. However, since the policy action only fails occasionally, this conservatism does not significantly impact overall performance and, in most scenarios, helps ensure safety. For static obstacles, we use each cast ray to determine the obstacle's center position and radius, setting their velocity to zero. For dynamic obstacles, we enclose them using one or multiple spheres.

IV. Result and Discussion

To evaluate the proposed framework, we present our training results under different configurations and conduct simulation and physical flight tests in various environments. The policy was trained in NVIDIA Isaac Sim on a NVIDIA GeForce RTX 4090 GPU for around 10 hours. The maximum velocity of the robot is set to 2.0 m/s. The simulation experiments are conducted on the RTX 4090 desktop, while computations for the physical flights are performed on our quadcopter's onboard computer (NVIDIA Jetson Orin NX). An Intel RealSense D435i camera is utilized for static and dynamic obstacle perception, and the LiDAR Inertial Odometry (LIO) algorithm [47] is adopted for accurate robot state estimation. The static and dynamic feature extractors use 3-layer convolutional neural networks, outputting embeddings of sizes 128 and 64, respectively. The policy network consists of a two-layer multi-layer perceptron with a PPO clip ratio of 0.1. The ADAM optimizer is used with a learning rate of 5×10^{-4}. The reward discounting factor is set to 0.99.

A. RL Training Results

Our framework utilizes a curriculum learning strategy for our RL policy training, as illustrated by the initial and final training environments shown in Fig. 6. Static obstacles are shown in red with a color gradient indicating height, while dynamic obstacles are depicted in green. Based on our observations, dynamic obstacles present a greater challenge during training than static obstacles. Therefore, we gradually increase the number of dynamic obstacles in the environment from 60 to 120, in increments of 20, once the navigation success rate exceeds 80% (defined as safely navigating from the start to the goal without collisions). To demonstrate the effectiveness of curriculum learning, Table I compares the highest navigation success rates achieved with and without curriculum learning, keeping the total training time constant for both scenarios. The table shows



(a) Initial Training Environment (b) Final Training Environment

Fig. 6. Visualization of robot training environments. The environment has a size of 50 m× 50 m, and the number of dynamic obstacles is gradually increased during training. (a) The initial environment contains 60 dynamic obstacles. (b) The final environment contains 120 dynamic obstacles.

TABLE I
Comparison of the Highest Navigation Success Rates During Training With and Without Using Curriculum Learning

Number of Obstacles	No Curr. Learning	Curr. Learning
static=350, dynamic=60	94.33%	94.33%
static=350, dynamic=80	74.51%	82.71%
static=350, dynamic=100	62.30%	80.96%
static=350, dynamic=120	54.98%	68.65%

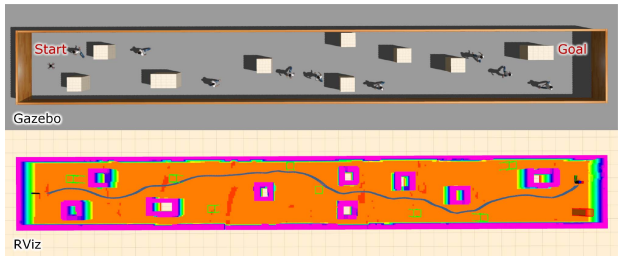


Fig. 7. Visualization of a safe navigation trajectory in a simulated corridor environment. The top image illustrates the environment in Gazebo, containing static obstacles and dynamic obstacles (pedestrians). The bottom image displays the environment map with the robot's navigation trajectory.

that as the number of dynamic obstacles increases, the highest navigation success rate without curriculum learning decreases more sharply compared to training with curriculum learning, highlighting its effectiveness. We stopped policy training at 120 dynamic obstacles and saved the best model trained with 100 dynamic obstacles, as it achieved a relatively high success rate (80.96%) under challenging conditions.

To show the importance of training with a larger number of robots, Fig. 8 compares the average RL training returns achieved with different numbers of robots deployed in training. The figure shows that training with more robots not only leads to faster convergence but also results in higher RL returns. Our experiments were conducted with 1024 robots, which utilized the maximum available GPU memory.

B. Simulation Experiments

We conducted simulation experiments in Gazebo to evaluate the proposed framework and demonstrate its sim-to-sim transferability. Quantitative tests were performed in indoor-like dynamic environments resembling real-world scenarios where the robot navigates safely around humans. Fig. 7 shows an example of a

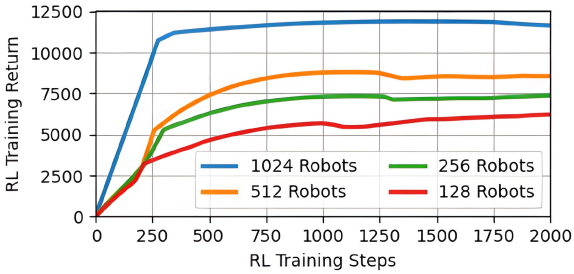


Fig. 8. Plot of the smoothed average training return curves. Training with a larger number of robots leads to faster convergence and higher returns.

TABLE II
BENCHMARK OF THE AVERAGE NUMBER OF COLLISIONS EVALUATED FROM 20 SAMPLE RUNS IN DIFFERENT TYPES OF ENVIRONMENTS

Average Collision Times Measurement in a 20m×40m Map			
Benchmarks	Static Env.	Dynamic Env.	Hybrid Env.
EGO [48]	0.45 (56.3%)	N/A	N/A
ViGO [20]	0.80 (100%)	3.15 (100%)	4.40 (100%)
Ours w/o Safe	0.95 (118.8%)	2.70 (85.7%)	4.60 (104.5%)
Ours (NavRL)	**0.65 (81.3%)**	**0.85 (27.0%)**	**2.10 (47.8%)**

safe trajectory. The results confirm that the robot successfully navigated without collisions.

To quantitatively evaluate the performance, we generate environments with high obstacle density, mirroring the conditions of our training environments, where both static and dynamic obstacles are randomly placed. Given the limited availability of open-source RL-based navigation benchmark algorithms, we compare our method with the popular optimization-based static planner [48] and a vision-aided planner [20] designed for dynamic environments. Additionally, we evaluate the performance of our framework with and without the safety shield to verify its effectiveness. Specifically, we test in three types of environments: static, purely dynamic, and hybrid environments. Each algorithm is run 20 times in each environment, and the average number of collisions per run is calculated as shown in Table II. The percentage value in Table II indicates the average collision times relative to the baseline method [20]. It is important to note that these environments are designed to test the limits of the algorithms and are significantly more complex than real-world scenarios. Therefore, the occurrence of collisions does not necessarily indicate that the algorithm is unsafe. Overall, our NavRL demonstrates the lowest number of collisions in dynamic and hybrid environments, while maintaining a comparable collision rate to the EGO planner in static environments. The table shows N/A for the EGO planner in dynamic and hybrid environments because its inefficient map updates cause it to get stuck when handling excessive noise from dynamic obstacles. In the experiments, ViGO fails to provide sufficiently reactive trajectories for collision avoidance when obstacles are in close proximity to the robot. In contrast, our method can generate more reactive controls that help the robot avoid collisions more efficiently. Comparing our framework with and without the safety shield, we found the safety shield consistently reduced collisions, particularly in dynamic environments, where it mitigates errors caused by the neural network's increased failure risk.

Fig. 9. Examples of physical flight tests. Our framework enables the robot's safe flight and navigation in the presence of static and dynamic obstacles.

TABLE III
THE RUNTIME OF EACH COMPONENT OF THE PROPOSED SYSTEM

System Modules	GeForce RTX 4090	Jetson Orin NX
Static Perception	8 ms	15 ms
Dynamic Perception	11 ms	27 ms
RL Policy Network	1 ms	7 ms
Safety Shield	2 ms	16 ms

C. Physical Flight Tests

To demonstrate sim-to-real transfer and safe navigation, we conducted physical flight experiments in various settings, as shown in Fig. 9. Static obstacles were placed in the environments, and several pedestrians were directed to walk toward the robot, requiring it to avoid collisions while reaching its goals. The results showed that the robot successfully avoided collisions and reached its destinations safely.

During the simulation and real-world flight experiments, we measured the computation time for each module in the proposed framework, as detailed in Table III. The table shows the runtimes on both the NVIDIA GeForce RTX 4090 and the onboard NVIDIA Jetson Orin NX computer. The static perception module completes in 8 ms on the RTX 4090 and 15 ms on the Orin NX, while the dynamic perception module takes 11 ms and 27 ms, respectively. The RL policy network runs in 1 ms on the RTX 4090 and 7 ms on the Orin NX, and the safety shield module operates in 2 ms and 16 ms. These measurements demonstrate that all modules are capable of real-time performance even on an onboard computer.

V. CONCLUSION AND FUTURE WORK

This letter presents a novel deep reinforcement learning framework, NavRL, designed to achieve safe flight in dynamic environments based on the Proximal Policy Optimization (PPO) algorithm. The framework uses tailored state and action representations to enable safe navigation in both static and dynamic environments, supporting effective zero-shot sim-to-sim and sim-to-real transfer. Besides, A safety shield based on the velocity obstacle concept mitigates failures from the black-box nature of neural networks. Additionally, a parallel training pipeline with NVIDIA Isaac Sim accelerates the learning process. Results from simulation and physical experiments verify the effectiveness of our approach in achieving safe navigation within dynamic environments. Future work will focus on improving and adapting this framework for deployment across various robotic platforms.

REFERENCES

[1] B. Zhou, Y. Zhang, X. Chen, and S. Shen, "FUEL: Fast UAV exploration using incremental frontier structure and hierarchical planning," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 779–786, Apr. 2021.

[2] S. H. Alsamhi et al., "UAV computing-assisted search and rescue mission framework for disaster and harsh environment mitigation," *Drones*, vol. 6, no. 7, 2022, Art. no. 154.

[3] Z. Xu, B. Chen, X. Zhan, Y. Xiu, C. Suzuki, and K. Shimada, "A vision-based autonomous UAV inspection framework for unknown tunnel construction sites with dynamic obstacles," *IEEE Robot. Automat. Lett.*, vol. 8, no. 8, pp. 4983–4990, Aug. 2023.

[4] M. Kulkarni and K. Alexis, "Task-driven compression for collision encoding based on depth images," in *Proc. Int. Symp. Vis. Comput.*, 2023, pp. 259–273.

[5] D. Hoeller, L. Wellhausen, F. Farshidian, and M. Hutter, "Learning a state representation and navigation in cluttered and dynamic environments," *IEEE Robot. Automat. Lett.*, vol. 6, no. 3, pp. 5081–5088, Jul. 2021.

[6] Y. Song, K. Shi, R. Penicka, and D. Scaramuzza, "Learning perception-aware agile flight in cluttered environments," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 1989–1995.

[7] D. Gandhi, L. Pinto, and A. Gupta, "Learning to fly by crashing," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 3948–3955.

[8] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield, "Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 4241–4247.

[9] V. Tolani, S. Bansal, A. Faust, and C. Tomlin, "Visual navigation among humans with optimal control as a supervisor," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 2288–2295, Apr. 2021.

[10] A. Ray, J. Achiam, and D. Amodei, "Benchmarking safe exploration in deep reinforcement learning," 2019, *arXiv:1910.01708*.

[11] B. Thananjeyan et al., "Recovery RL: Safe reinforcement learning with learned recovery zones," *IEEE Robot. Automat. Lett.*, vol. 6, no. 3, pp. 4915–4922, Jul. 2021.

[12] K.-C. Hsu, A. Z. Ren, D. P. Nguyen, A. Majumdar, and J. F. Fisac, "Sim-to-Lab-to-Real: Safe reinforcement learning with shielding and generalization guarantees," *Artif. Intell.*, vol. 314, 2023, Art. no. 103811.

[13] N. Kochdumper, H. Krasowski, X. Wang, S. Bak, and M. Althoff, "Provably safe reinforcement learning via action projection using reachability analysis and polynomial zonotopes," *IEEE Open J. Control Syst.*, vol. 2, pp. 79–92, 2023.

[14] F. Sadeghi and S. Levine, "CAD2RL: Real single-image flight without a single real image," in *Proc. Robot.: Sci. Syst.*, Cambridge, Massachusetts, Jul. 2017, doi: 10.15607/RSS.2017.XIII.034.

[15] L. Xie, S. Wang, A. Markham, and N. Trigoni, "Towards monocular vision based obstacle avoidance through deep reinforcement learning," 2017, *arXiv:1706.09829*.

[16] R. Brilli, M. Legittimo, F. Crocetti, M. Leomanni, M. L. Fravolini, and G. Costante, "Monocular reactive collision avoidance for MAV teleoperation with deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 12535–12541.

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

[18] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *Int. J. Robot. Res.*, vol. 17, no. 7, pp. 760–772, 1998.

[19] Y. Wang, J. Ji, Q. Wang, C. Xu, and F. Gao, "Autonomous flights in dynamic environments with onboard vision," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 1966–1973.

[20] Z. Xu, Y. Xiu, X. Zhan, B. Chen, and K. Shimada, "Vision-aided UAV navigation and dynamic obstacle avoidance using gradient-based B-spline trajectory optimization," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 1214–1220.

[21] G. Chen, P. Peng, P. Zhang, and W. Dong, "Risk-aware trajectory sampling for quadrotor obstacle avoidance in dynamic environments," *IEEE Trans. Ind. Electron.*, vol. 70, no. 12, pp. 12606–12615, Dec. 2023.

[22] L. Tai, S. Li, and M. Liu, "A deep-network solution towards model-less obstacle avoidance," in *Proc. IEEE/RSJ Int. Conf. Intell. robots Syst.*, 2016, pp. 2759–2764.

[23] R. P. Padhy, S. Verma, S. Ahmad, S. K. Choudhury, and P. K. Sa, "Deep neural network for autonomous UAV navigation in indoor corridor environments," *Procedia Comput. Sci.*, vol. 133, pp. 643–650, 2018.

[24] A. Loquercio, A. I. Maqueda, C. R. Del-Blanco, and D. Scaramuzza, "DroNet: Learning to fly by driving," *IEEE Robot. Automat. Lett.*, vol. 3, no. 2, pp. 1088–1095, Apr. 2018.

[25] S. Jung, S. Hwang, H. Shin, and D. H. Shim, "Perception, guidance, and navigation for indoor autonomous drone racing using deep learning," *IEEE Robot. Automat. Lett.*, vol. 3, no. 3, pp. 2539–2544, Jul. 2018.

[26] S. Lv, Y. Gao, J. Che, and Q. Quan, "Autonomous drone racing: Time-optimal spatial iterative learning control within a virtual tube," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 3197–3203.

[27] D. Shah et al., "ViNT: A foundation model for visual navigation," in *Proc. 7th Annu. Conf. Robot Learn.*, 2023. [Online]. Available: https://arxiv.org/abs/2306.14846

[28] D. Shah, A. Sridhar, A. Bhorkar, N. Hirose, and S. Levine, "GNM: A general navigation model to drive any robot," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 7226–7233.

[29] A. Sridhar, D. Shah, C. Glossop, and S. Levine, "NoMAD: Goal masked diffusion policies for navigation and exploration," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2024, pp. 63–70.

[30] N. Simon and A. Majumdar, "MonoNav: MAV navigation via monocular depth estimation and reconstruction," in *Proc. Int. Symp. Exp. Robot.*, 2023, pp. 415–426.

[31] S. F. Bhat, R. Birkl, D. Wofk, P. Wonka, and M. Müller, "ZoeDepth: Zero-shot transfer by combining relative and metric depth," 2023, *arXiv:2302.12288*.

[32] P. Roth, J. Nubert, F. Yang, M. Mittal, and M. Hutter, "ViPlanner: Visual semantic imperative learning for local navigation," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2024, pp. 5243–5249.

[33] A. Singla, S. Padakandla, and S. Bhatnagar, "Memory-based deep reinforcement learning for obstacle avoidance in UAV with limited environment knowledge," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 1, pp. 107–118, Jan. 2021.

[34] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 285–292.

[35] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 1343–1350.

[36] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 3052–3059.

[37] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, no. 7976, pp. 982–987, 2023.

[38] L. Bauersfeld, E. Kaufmann, and D. Scaramuzza, "User-conditioned neural control policies for mobile robotics," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 1342–1348.

[39] A. Romero, Y. Song, and D. Scaramuzza, "Actor-critic model predictive control," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2024, pp. 14777–14784.

[40] J. Xing, L. Bauersfeld, Y. Song, C. Xing, and D. Scaramuzza, "Contrastive learning for enhancing robust scene transfer in vision-based agile flight," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2024, pp. 5330–5337.

[41] B. Xu, F. Gao, C. Yu, R. Zhang, Y. Wu, and Y. Wang, "OmniDrones: An efficient and flexible platform for reinforcement learning in drone control," *IEEE Robot. Automat. Lett.*, vol. 9, no. 3, pp. 2838–2844, Mar. 2024.

[42] T. He, C. Zhang, W. Xiao, G. He, C. Liu, and G. Shi, "Agile but safe: Learning collision-free high-speed legged locomotion," in *Proc. Robot.: Sci. Syst.*, Delft, Netherlands, Jul. 2024, doi: 10.15607/RSS.2024.XX.059.

[43] Z. Xu, X. Zhan, Y. Xiu, C. Suzuki, and K. Shimada, "Onboard dynamic-object detection and tracking for autonomous robot navigation with RGB-D camera," *IEEE Robot. Automat. Lett.*, vol. 9, no. 1, pp. 651–658, Jan. 2024.

[44] H. Oleynikova, D. Honegger, and M. Pollefeys, "Reactive avoidance using embedded stereo vision for MAV flight," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2015, pp. 50–56.

[45] P.-W. Chou, D. Maturana, and S. Scherer, "Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 834–843.

[46] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Proc. Robot. Res.: 14th Int. Symp. ISRR*, 2011, pp. 3–19.

[47] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "FAST-LIO2: Fast direct LiDAR-inertial odometry," *IEEE Trans. Robot.*, vol. 38, no. 4, pp. 2053–2073, Aug. 2022.

[48] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao, "EGO-planner: An ESDF-free gradient-based local planner for quadrotors," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 478–485, Apr. 2021.