IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 14 September 2024, accepted 1 October 2024, date of publication 4 October 2024, date of current version 26 November 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3473935

RESEARCH ARTICLE

Real-Time Parameter Control for Trajectory Generation Using Reinforcement Learning With Zero-Shot Sim-to-Real Transfer

CHANG-HUN JI[1], GYEONGHUN LIM[2], YOUN-HEE HAN[1], (Member, IEEE), AND SUNGTAE MOON[2]

[1]Department of Future Convergence Engineering, Korea University of Technology and Education, Cheonan-si 31253, South Korea
[2]Department of Intelligent System and Robotics, Chungbuk National University, Cheongju-si 28644, South Korea

Corresponding authors: Youn-Hee Han (yhhan@koreatech.ac.kr) and Sungtae Moon (stmoon@cbnu.ac.kr)

This work was supported in part by the Unmanned Vehicle Advanced Research Center (UVARC) funded by the Ministry of Science and Information and Communication Technology (MSIT) of Republic of Korea (40%) under Grant 2023M3C1C1A01098416; in part by the Innovative Human Resource Development for Local Intellectualization Program through the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant funded by Korea Government (MSIT, 30%) under Grant IITP-2024-2020-0-01462; and in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (30%) under Grant 2018R1A6A1A03025526.

ABSTRACT Research on trajectory generation algorithms for unmanned ground vehicles (UGVs) has been actively conducted due to the rapid increase in their use across various fields. Trajectory generation for UGVs requires a high level of precision, as various parameters determine the trajectory's efficiency and safety. Notably, maximum velocity and acceleration are critical factors impacting trajectory performance. In this paper, we propose a novel algorithm that utilizes reinforcement learning to determine the optimal maximum velocity and acceleration in real-time within dynamic environments. The proposed algorithm overcomes the limitations of traditional fixed parameter settings by determining parameters through real-time environmental adaptation. Furthermore, we also propose a PX4-ROS2 based reinforcement learning framework for achieving stable zero-shot sim-to-real transfer. Experimental results in simulation and real-world environments show that the proposed method significantly improves trajectory safety and efficiency while also demonstrating excellent adaptability to changing environments. Furthermore, validation through identical experimental results in both simulation and real-world environments confirms a stable zero-shot sim-to-real transfer.

INDEX TERMS Trajectory generation, reinforcement learning, real-time parameter optimization, sim-to-real.

I. INTRODUCTION

Recently, the use of unmanned ground vehicles (UGVs) has increased rapidly in various fields, and research on its trajectory generation algorithms has been actively conducted. Trajectory generation for UGV requires high precision beyond simply setting a path from origin to destination. This precision is influenced by numerous parameters that determine the trajectory's efficiency and safety. Among these parameters, we argue that the maximum velocity v_{max} and acceleration a_{max} of a UGV significantly impact the performance of the generated trajectories, as well as the maneuverability and safety of the UGV. Therefore, it is crucial to set v_{max} and a_{max} to optimal values in the trajectory generation algorithm. For instance, setting v_{max} too low may result in unnecessarily prolonged travel times to the target point. Conversely, if v_{max} is set too high, control difficulties may arise, thereby increasing the risk of crashes. Similarly, a_{max} can lead to comparable issues. Insufficient a_{max} may result in poor maneuverability, whereas excessive acceleration can compromise the stability of UGV.

However, optimizing these parameters in a dynamic and real-time environment is extremely challenging. Traditional trajectory generation algorithms rely on fixed and preset v_{max}

The associate editor coordinating the review of this manuscript and approving it for publication was Jiachen Yang.

and a_{max}, which lack adaptability to environmental changes and lead to inefficient and unsafe trajectories [1], [2], [3], [4], [5]. In this paper, to address these issues, we propose a novel algorithm that utilizes reinforcement learning (RL) [6] to determine the optimal v_{max} and a_{max} for trajectory generation in real-time in dynamic environments. Unlike traditional fixed parameterization, an RL agent can adaptively learn and determine optimal values without relying on preset data, enabling real-time learning and adjustment. Moreover, this approach effectively accounts for the surrounding environment and overcomes the limitations of traditional trajectory generation algorithms.

On the other hand, RL requires a large amount of data for training due to its trial-and-error learning processes. Training in the real-world can be significantly more costly than in simulations, especially when a large amount of data is needed. Therefore, we train RL deep learning models using high-fidelity simulations to gather the necessary data efficiently and cost-effectively before deploying them in real-world environments. However, the inherent differences between simulations and the real-world create a reality gap that reduces the stability of sim-to-real transfer. For this reason, various research has been conducted to achieve stable sim-to-real transfer in RL [7], [8]. Most sim-to-real research in RL has focused on using real-world data for additional training [9], [10], [11] or employing additional models [12], [13], [14]. However, when the experimental cost in real-world environments is high, additional training with real-world data becomes impractical. Moreover, employing additional models can increase complexity, potentially reducing training performance or forcing further training.

In this paper, we propose a framework, called the PX4-ROS2 based framework, which uses PX4 and Robot Operating System 2 (ROS2) to incorporate real-world operation processes into simulation training. The PX4-ROS2 based framework is designed to reduce the reality gap without additional models or real-world data. Additionally, we introduce methods to enhance the RL agent's generalization performance, which is crucial for effective sim-to-real transfer. These methods allow for reliable zero-shot sim-to-real transfer without requiring further training or tuning in the real-world. Zero-shot sim-to-real transfer refers to the capability of applying a model trained entirely in simulation directly to real-world scenarios without any additional training or fine-tuning.

Through comparative experiments, we compare the performance of trajectories generated by our proposed method with those generated by fixed parameterization. These experiments are conducted in both simulated and real-world environments with zero-shot sim-to-real transfer. In the simulation experiments, we analyze the trajectory generation performance of our proposed method and an existing trajectory generation method across various scenarios. This analysis confirms that the proposed method significantly improves trajectory safety and efficiency by optimizing v_{max} and a_{max} in real-time. We also validate our algorithm in real-world environments with a different obstacle configuration completely from the simulated environment. The validation results show that the proposed method can reliably generate optimal trajectories in real-world settings with unseen obstacle configurations, which indicates that our proposed method has strong generalization performance.

Finally, we configure the simulation environment to resemble the real experimental environment and conducted experiments to verify that the proposed method operates consistently in both environments. The comparison results indicate that similar velocities of the UGV are observed in both the real-world environment and the simulation. This indicates that the proposed method has effectively learned a stable model for sim-to-real transfer through the PX4-ROS2 based RL framework.

The contributions of this paper are as follows: 1) We propose a novel RL-based algorithm that optimizes v_{max} and a_{max} for real-time UGV trajectory generation in dynamic environments, and 2) We introduce a PX4-ROS2 based RL framework to enable stable zero-shot sim-to-real transfer without additional training or tuning. Through extensive experiments, we demonstrate that the proposed RL-based method outperforms an existing method with fixed parameter settings in terms of stability, efficiency, and adaptability to real-time changes.

The rest of this paper is organized as follows. Section II reviews related works on trajectory generation for UGVs. Section III covers the preliminaries, including the RL framework used. In Section IV, we describe the proposed method and its design. Section V the experimental setup and results. Finally, Section VI discusses the findings and their implications, and also concludes the paper.

II. RELATED WORKS

Trajectory generation algorithms for UGVs have been actively studied. Zhou et al. [15] proposed a strategy combining the artificial fish swarm algorithm for global trajectory planning with a trial-based forward search for local adjustments to ensure UGVs navigate safely around unforeseen obstacles. Ren et al. [16] presented an optimal path planning and speed control strategy for UGVs, integrating dynamic programming and model predictive control. Brito et al. [17] proposed a model predictive contouring control for handling obstacles in unstructured environments. Their approach demonstrated superior real-time collision avoidance in UGV experiments.

Research on applying unmanned aerial vehicle (UAV) trajectory generation algorithms to UGVs has also achieved significant outcomes. Chen et al. [18] utilized the polyhedral representations and decentralized planning approach of MADER [3] to develop a trajectory generation algorithm for UGVs. Liu et al. [19] developed a UGV trajectory generation algorithm influenced by the gradient calculation method of EGO-Planner [2] and conducted performance comparisons between their algorithm and EGO-Planner

applied to UGVs. Moreover, Li et al. [20] proposed a collaborative air-ground framework for perception-limited UGVs' navigation and evaluated its performance by directly applying EGO-Swarm [21], which is developed using EGO-Planner, to UGVs.

However, these trajectory generation algorithms employ fixed parameters without accounting for the surrounding environment, making it challenging to generate optimal paths in real-world settings with rapidly changing obstacles. This limitation can lead to suboptimal performance if the parameters do not adapt to the environment.

In contrast, RL is a type of machine learning in which an agent interacts with its environment [6]. The agent performs actions within this environment, receives feedback in the form of rewards or penalties, and learns to adjust its behavior to maximize cumulative rewards over time. This learning process involves exploring various actions and exploiting the knowledge gained to optimize future decisions. A growing number of studies utilize the benefits of RL to control unmanned vehicles (UVs) such as UGVs and UAVs, and demonstrate its potential to improve UV trajectory generation. For instance, a recent study applied RL to optimize power allocation and UAV service zones in emergency communication networks [22]. Similarly, RL has been employed in UAV trajectory design and frequency band allocation, demonstrating not only energy efficiency but also fairness in communication systems [23].

Moreover, the high combinatorial optimization capabilities of RL allow for real-time parameter control across various fields. Specifically, Karafotias et al. [24] introduced an RL-based controller that dynamically adjusts parameters in evolutionary algorithms, enhancing solution quality without additional resources or effort. Similarly, Chen et al. [25] proposed a method using RL to optimize parameters in genetic algorithms, significantly improving efficiency and solution quality in the flexible job-shop scheduling problem. Leveraging these capabilities, RL can be utilized to optimize various parameters in trajectory generation algorithms.

However, a significant drawback of RL is its reliance on trial and error, which makes it unreliable in complex environments that require extensive training. Due to this limitation, most RL-based UV control research has been validated in simple grid environments [26], [27], [28]. Additionally, models trained in simulations are often not transferred effectively to real-world scenarios [17], [29]. Even when sim-to-real transfer is achieved, the results are only utilized as reference paths for model predictive control and do not account for dynamic environments [30], [31].

Sim-to-real transfer typically involves tuning in real-world settings after simulation training or incorporating real-world data during training to reduce the sim-to-real gap [9], [10], [11]. However, this approach can be impractical when the cost of real-world experimentation is high. Therefore, zero-shot sim-to-real transfer, which does not require additional real-world training or data, has been explored through various approaches. Nonetheless, most zero-shot sim-to-real transfers have been conducted on robotic arms in relatively constrained environments, and those involving moving vehicles generally assume short-distance travel [13], [32], [33], [34].

III. PRELIMINARIES

A. MARKOV DECISION PROCESS

The Markov decision process (MDP) provides a mathematical framework for modeling the problem of an RL agent interacting with its environment and learning optimal behavior [35]. An MDP can be defined as a tuple (S, U, R, P, γ). S is the state space representing all possible configurations of the environment. U is the action space available to the RL agent. $P(s'|s, u)$ is the state-transition probability function, representing the probability of transitioning to the next state s' given action u in state s. $R(s, u, s')$ is the reward function, providing feedback to the agent in the form of a reward r. γ is the discount factor that balances the importance of immediate and future rewards, where a value closer to 1 places greater importance on future rewards. An RL agent learns a policy $\pi(u|s)$ to maximize its expected cumulative reward over time. The policy $\pi(u|s)$ is a probability distribution over an action u that the RL agent decides upon in a given state s. For each time step t in an episode, the optimal policy π is defined as follows:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{u_t \sim \pi} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right], \tag{1}$$

where r_t represents the reward at time step t and T is the episode length.

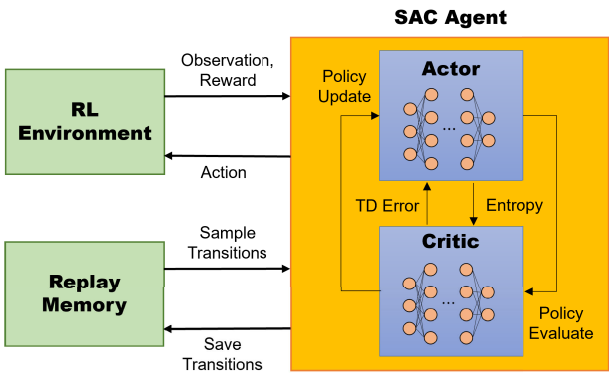


FIGURE 1. The SAC agent training process.

In many applications where decisions need to be made based on partial and noisy observations, un agent may not be able to fully observe the entire state of the environment. To address this, the MDP framework is extended to partially observed Markov decision process (POMDP). A POMDP can be defined as a tuple $(S, U, R, P, \Omega, O, \gamma)$. Ω represents the observation space, which includes all possible observations the agent can perceive. $O(o|s, u)$ is the observation probability function, indicating the probability of observing o given that action u was taken in state s. On the other hand, S, U,

P, R, and γ are defined the same way as in MDPs. In a POMDP, an agent cannot directly access the state $s \in S$. Instead, it receives observations $o \in \Omega$ that provide partial information about the state. The agent's goal is to choose actions that maximize its cumulative reward, similar to a MDP. In this paper, we model the task as a POMDP, which allows us to use RL to solve real-time parameter optimization problems in an unknown environment.

B. SOFT ACTOR CRITIC

The Soft Actor-Critic (SAC) algorithm is one of the most popular RL algorithms. It combines off-policy learning with entropy regularization to enhance both sample efficiency and exploration performance [36]. The SAC algorithm employs an actor-critic architecture, where the actor updates a policy $\pi_\phi(u|o)$ for selecting an action u from an observation o, and the critic updates a value function $Q_\omega(o, u)$ that estimates the state-action value. In this architecture, ϕ and ω represent the model parameters of the policy and the value function, respectively. These two components work together to maximize the agent's long-term reward.

A replay buffer $\mathcal{B}$ is used to store transitions that the agent experiences during training. This replay buffer allows the SAC algorithm to break the correlation between consecutive transitions by sampling random batches of transitions for training, which improves learning stability and efficiency. The buffer stores tuples of observation o, action u, reward r, and next observation o', which are used to update the policy and value functions.

On the other hand, the entropy for an observation o is defined as follows.

$$\mathcal{H}(o) = \mathbb{E}_{U\sim\pi_\phi(\cdot|o)}[-\log \pi_\phi(u|o)]. \tag{2}$$

In SAC, the entropy term is incorporated into the policy's probability distribution, which encourages an SAC agent to explore various behaviors under uncertainty. This algorithm excels in high-dimensional continuous state and action spaces due to its stable learning process and rapid convergence.

For a tuple (o, u, r, o') sampled from $\mathcal{B}$, the entropy-regularized critic's loss function is defined as follows.

$$J_Q(\omega) = \mathbb{E}_{u'\sim\pi_\phi(\cdot|o')}\Bigg[\frac{1}{2}\Bigg(Q_{\omega_i}(o, u) - \Bigg(r + \gamma \min_{j=1,2} Q_{\bar{\omega}_j}(o', u') - \alpha \log \pi_\phi(u'|o')\Bigg)\Bigg)^2\Bigg] \quad \text{for } i = 1, 2 \tag{3}$$

where $Q_{\bar{\omega}_i}(o', u')$ represents the ith target value function evaluated at the next observation o' and the sampled next action u', and α is the temperature parameter that determines the relative importance of the entropy term versus the reward, and thus controls the stochasticity of the optimal policy. The actor is then updated by maximizing the following entropy-regularized objective function:

$$J_\pi(\phi) = \mathbb{E}_{u'\sim\pi_\phi(\cdot|o')}\Big[\min_{j=1,2} Q_{\omega_j}(o, u) - \alpha \log \pi_\phi(u'|o')\Big] \tag{4}$$

In addition, the SAC agent automatically adjusts the temperature parameter α during training to maintain an appropriate exploration-exploitation balance. To achieve this, it sets an entropy target value $\bar{\mathcal{H}}$ and adjusts α to align closely with the actual entropy target value by minimizing the following loss.

$$J(\alpha) = \mathbb{E}\Big[-\alpha \log \pi_\phi(u|o) - \alpha\bar{\mathcal{H}}\Big] \tag{5}$$

Fig. 1 illustrates the overall learning structure of the SAC agent. The SAC agent learns through continuous interaction with the RL environment. The trajectories generated through this interaction are stored in replay memory $\mathcal{B}$ and sampled during learning. While the critic is updated by the entropy-regularized critic's loss function (Eq. (3)), it aids the actor by providing policy evaluation to facilitate the actor's policy update. The actor, in turn, contributes entropy when updating the critic to encourage exploration, and the critic is utilized to generate the actor's objective (Eq. (4)).

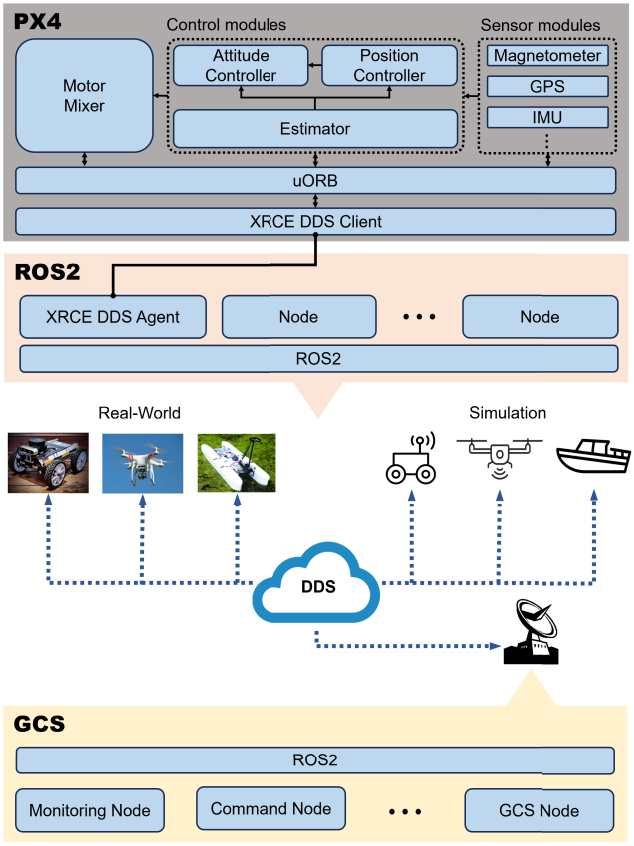


FIGURE 2. The PX4-ROS2 architecture.

C. PX4-ROS2

PX4 is an open-source control software extensively utilized across various UAV platforms [37]. While it has primarily been used for controlling UAVs and other types of drones, its robustness, flexibility, and modularity have also led to its application in many UV systems such as UGVs and unmanned underwater vehicles [38], [39]. Similarly, ROS2 is an open-source framework designed to support

TABLE 1. The PX4 configuration settings for UGV operation.

Parameters	Values	Inference
GND_L1_PERIOD	1.0	Tracking point ahead
GND_L1_DAMPING	0.75	Damping for L1 control
GND_MAX_ANG	0.7854	Max turn angle
GND_WHEEL_BASE	0.31	Distance front to rear axle
COM_RCL_EXCEPT	Offboard	Remote control loss exceptions
GND_SP_CTRL_MODE	GPS speed	Control mode for speed

TABLE 2. Notation used in POMDP.

Notation	Description
v_t^{max}	The maximum velocity at time step t
a_t^{max}	The maximum acceleration at time step t
$W \times H$	The size of environment
o_t^{map}	A top view of the UGV's surrounding at time step t
o_t^{path}	The UGV direction and path to the target at time step t
$w_o \times h_o$	The size of o_t^{map} and o_t^{path}
o_t	The observation at time step t, $o_t = (o_t^{map}, o_t^{path})$
u_t	The action at time step t, $u_t = (v_t^{max}, a_t^{max})$
r_t	The reward at time step t, defined by Eq. (6)

the development of complex robotic systems [40]. The integration of PX4 with ROS2 (PX4-ROS2) leverages the strengths of both platforms and creates a highly flexible and scalable solution for various robotic applications.

The PX4-ROS2 architecture, as depicted in Fig. 2, highlights the detailed interaction between various modules within the PX4 system. The control modules include the Attitude controller, Position controller, and Estimator. The Estimator receives raw data from the sensor module and uses an extended Kalman filter to refine it into precise dynamic states such as position and attitude. These refined dynamic states are then provided to the Attitude controller and Position controller. The Attitude controller and Position controller utilize the dynamic states to calculate the thrust and moment required to execute the current mission. These calculations are subsequently provided to the motor mixer module. The motor mixer module translates these into PWM signals and transmits them to the motors. In PX4, modules interact with each other using the micro object request broker (uORB), which employs a message-driven publish-subscribe architecture.

ROS2 is also highly modular, which allows flexible configuration and integration of various components. The high modularity of ROS2 allows it to be applied across various platforms, including simulation and real-world environments. This approach enables us to use ROS2 to run both simulation and real-world environments with the same code, making it easier to validate the algorithm. Moreover, ROS2 incorporates a data distribution service (DDS) architecture [41], which enables seamless communication among multiple heterogeneous UVs. These attributes facilitate the operation of these UVs through the control processes of PX4 and the DDS architecture of ROS2. At this time, each unmanned vehicle is controlled using the ground control system (GCS).

Given that both ROS2 and PX4 utilize a publish-subscribe architecture, this allows for seamless integration between PX4 and ROS2. However, integration of PX4 with ROS2 presents challenges due to PX4's insufficient computing power to incorporate ROS2 directly. Therefore, the integration needs to be facilitated through tools such as the XRCE-DDS (Data Distribution Service for eXtremely Resource-Constrained Environments) middleware [42]. The XRCE-DDS client runs on PX4 to handle the publication and subscription of PX4 messages, while the XRCE-DDS agent runs on companion computers. This agent acts as a proxy and converts PX4 data into ROS2 messages and ROS2 messages back into the PX4 internal message format. This bidirectional data translation ensures correct data exchange between both systems.

In this paper, we construct our autonomous UGV navigation system using the PX4-ROS2 framework. The PX4-ROS2 system integrates a variety of components to manage UGV operations. These components are crucial for ensuring the UGV can navigate dynamic environments effectively, as they enable real-time communication between the UGV's sensors, controllers, and actuators.

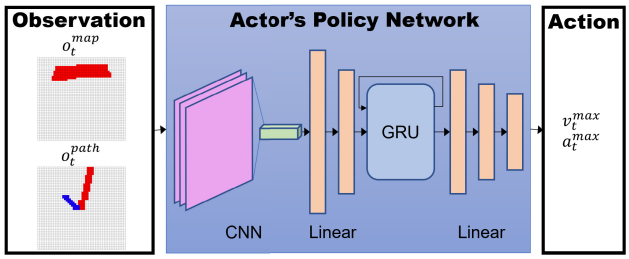


FIGURE 3. The actor's policy network architecture.

In our system, the PX4 configuration settings define the UGV's behavior across various scenarios. These settings are essential as they systematically constrain and regulate the UGV's physical movements, ensuring safe and reliable operation under varying conditions. Within the PX4 configuration settings, the RL agent controls trajectory generation algorithm's parameters in real-time to optimize the UGV's trajectory. Table 1 lists several crucial PX4 configurations for the UGV.

IV. METHODOLOGY

A. POMDP FORMULATION

Traditional trajectory generation algorithms rely on fixed parameter settings, which fail to adapt to continuously changing environmental conditions. This challenge prevents the algorithm from generating truly optimal paths, as the parameters need to be adjusted dynamically in response to the environment. To address this challenge, we formulate the problem of controlling the v_{max} and a_{max} of the trajectory generation algorithm in real-time as a POMDP in this section. Notations used in the POMDP are shown in Table 2.

TABLE 3. The specifications of the actor's policy network.

Type	Details
Conv2D + ReLU	2 channels, 32 filters, kernel 8x8, stride 4
Conv2D + ReLU	32 channels, 64 filters, kernel 4x4, stride 2
Conv2D + ReLU	64 channels, 64 filters, kernel 3x3, stride 1
Linear + ReLU	Input: 3136, Output: 512
Linear + ReLU	Input: 512, Output: 256
GRU	Input: 256, Hidden Size: 512, Layers: 1
Linear + ReLU	Input: 512, Output: 256
Linear + ReLU	Input: 256, Output: 128
Linear + Tanh	Input: 128, Output: 2

1) OBSERVATION AND ACTION

In this paper, the UGV performs a predefined mission within a fixed $W \times H$ square map. The observation of the RL agent provides information about the UGV's surroundings within a size of $w_o \times h_o$, where $w_o \ll W$ and $h_o \ll H$. This observation is composed of two components. The first component $o_t^{map} \in \mathbb{R}^{w_o \times h_o}$ represents the surrounding environments of the UGV, while the second component $o_t^{path} \in \mathbb{R}^{w_o \times h_o}$ represents the straight path from the UGV to the target and the direction of the UGV.

Specifically, o_t^{map} is defined as a bird's-eye view of the UGV's surrounding environment. The current position of the UGV is always fixed at the center of o_t^{map}. This visually clarifies that the surrounding environment dynamically changes as the UGV moves. In o_t^{map}, obstacles are represented using binary values: a location is marked with 1 if an obstacle is present and 0 if it is clear. This binary representation helps the RL agent efficiently and rapidly understand the complex details of the surrounding environment. Through o_t^{map}, the RL agent can recognize and respond to various obstacles and complex terrain changes in the UGV's surrounding environment in real-time.

Meanwhile, o_t^{path} is defined by both the direction the UGV should go and the direction it is currently traveling. It clearly visualizes the relationship between the UGV's current heading and the target point. Specifically, the UGV's position is fixed at the center of o_t^{path}, just like in o_t^{map}. A straight path between the UGV and the target is represented by 1, helping the agent visually recognize the best path for the UGV to reach its destination. This allows the agent to easily ascertain the direction the UGV should take to reach the target point. Additionally, the straight path between the UGV's expected position after one second and the current position is represented by -1, enabling the agent to recognize the UGV's current heading. All other elements are represented by zeros. As a result, the RL agent can know how much the current heading of the UGV deviates from the target direction. This information allows the RL agent to predict the yaw axis rotation of the UGV.

Finally, the RL agent extracts $u_t = (v_t^{max}, a_t^{max}) \in \mathbb{R}^2$ for the trajectory generation algorithm as actions to maximize the cumulative reward from the given observation. As shown in Fig. 3, the actor's policy network processes the observations to determine the actions v_t^{max} and a_t^{max}. It consists of several layers and components. The convolutional neural network (CNN) [43] processes the input observation o_t, extracting spatial features from the input data. The fully connected (FC) layers then transform these features through multiple layers to capture complex relationships within the input. The gated recurrent unit (GRU) [44] processes the sequential hidden data, maintaining temporal dependencies and helping the actor understand the sequence of observations over time. Additional FC layers further refine the processed input to output the final action values. To ensure v_{max} and a_{max} within a fixed range, the final layer of the actor's policy network utilizes the Tanh activation function. The specifications of the actor's policy network layers are presented in Table 3.

2) REWARD FUNCTION

We aim to intuitively reduce collisions and minimize the time taken to reach the target point through the reward function. Our reward function is defined as follows:

$$r_t = \begin{cases} 1 & \text{if the target is reached,} \\ -1 & \text{if a collision occurrs,} \\ -1 & \text{if the episode max step is exceeded,} \\ -0.1 & \text{otherwise.} \end{cases} \tag{6}$$

As shown in Eq. (6), our reward function is simple, and expert knowledge is excluded in its definition. A complex reward scheme can hinder the agent's learning and cause it to optimize for specific scenarios while struggling to adapt to new ones. Therefore, a simple and intuitive reward scheme is essential, as it allows our RL agent to learn broadly applicable action patterns and perform consistently across different scenarios.

Consequently, the optimization target is to adjust the policy π to maximize the cumulative reward as defined in Eq. (1), using the rewards from Eq. (6) over an episode. By optimizing this policy, the agent learns to make decisions that enhance safety and efficiency, resulting in optimal trajectory generation in dynamic environments.

B. POLICY TRAINING

A key advantage of RL models with good generalization performance is their ability to adapt to new situations not encountered during training. This capability is crucial when the models encounter unexpected factors that were not present during training in the simulation. Therefore, a focus on enhancing generalization performance directly correlates with better sim-to-real transfer performance. In this section, to achieve this, we highlight two key components: curriculum-based random training map and recurrent neural networks.

1) CURRICULUM-BASED RANDOM TRAINING MAP

To enhance the generalization performance of RL agents in different environments, we utilize curriculum-based random training maps instead of fixed training maps. For every

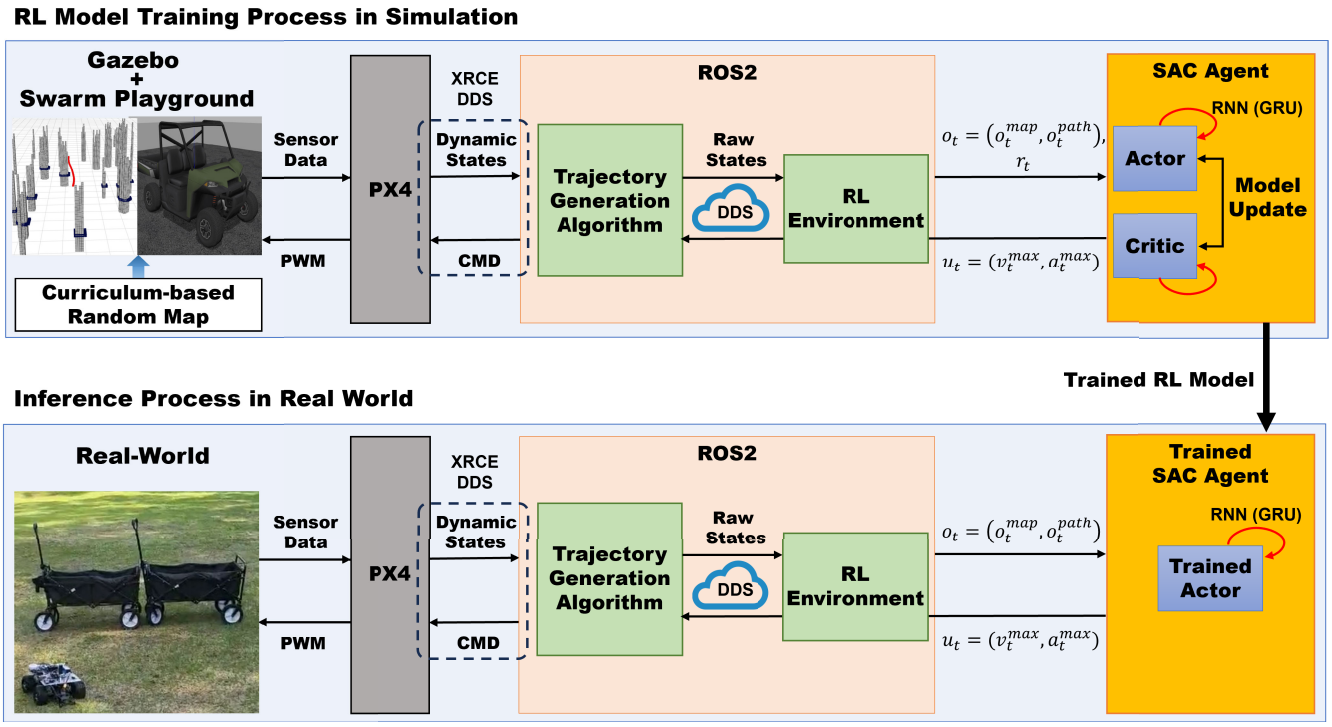


FIGURE 4. The PX4-ROS2 based RL framework.

episode, we adjust the difficulty of the training maps for curriculum learning to enable the RL agent to learn from easier maps initially. The difficulty of the maps was adjusted by varying the number of obstacles. Let n_i be the number of obstacles in the ith episode of the training phase. It is defined as follows:

$$n_i = \min\left(\lceil n_0 + k \cdot i \rceil, n_{max}\right), \tag{7}$$

where n_0 is the initial number of obstacles, $k \in \mathbb{R}$ is a constant that determines the growth rate of the number of obstacles, and n_{max} is the maximum number of obstacles. For the random training map in the ith episode, diverse obstacles are generated at random locations according to a uniform distribution $\mathcal{U}$ within a fixed $W \times H$ square map. In other words, the position of the jth obstacle is given by $obs_j = (x_j, y_j)$, where $(x_j, y_j) \sim (\mathcal{U}(0, W-1), \mathcal{U}(0, H-1))$.

This approach ensures that the RL agent encounters a wide range of configurations during training, which is critical for achieving robust generalization performance. Curriculum-based random training maps enable the RL agent to encounter diverse scenarios during training and improve adaptation and robustness in new environments.

2) RECURRENT NEURAL NETWORKS

To enhance the generalization performance of the RL agent, we incorporate a recurrent neural network (RNN) model, specifically a GRU, into the agent's policy and critic models. The GRU model can effectively capture the complex relationships and temporal dependencies within the sequential pairs of observation and action, leading to more efficient and adaptive trajectory generation. This is particularly important in our complex POMDP environments, where the RL agent cannot immediately access all state information and must infer the current situation from past data [45].

Moreover, when the RL agent for POMDP incorporates GRU, it can effectively apply knowledge learned in simulation to real-world environments and adapt to new situations. This capability helps the agent maintain consistent performance across various scenarios and improves sim-to-real transfer performance.

C. PX4-ROS2 BASED RL FRAMEWORK

In this paper, we propose a novel approach using RL to optimize the parameters of a trajectory generation algorithm in real-time within a dynamic environment. To validate this approach in the real-world, we propose a PX4-ROS2 based RL framework to ensure stable sim-to-real transfer. The proposed framework unifies behavior between simulation and real-world environments, and allows models trained in simulation to perform effectively in real-world conditions.

The architecture of PX4-ROS2 based RL framework is presented in Fig. 4. The proposed framework uses Gazebo [46] and Swarm Playground [21] to construct the simulation environment. Specifically, obstacle generation and detection are handled using Swarm Playground, while the physics-based simulation is conducted using Gazebo. In the simulation, the obstacle information is perceived through Swarm Playground, and various sensor data, excluding obstacle information, are acquired through Gazebo. Swarm

Algorithm 1 Proposed SAC Training Algorithm

T: trajectory generation algorithm
Γ_t: trajectory at time step t
d_t: dynamic states at time step t
$\mathcal{B}$: replay buffer
π_ϕ: SAC actor
Q_{ω_i}: SAC critics ($i = 1, 2$)
$Q_{\bar{\omega}_i}$: SAC target critics ($i = 1, 2$)
α: temperature parameter
$\lambda_Q, \lambda_\pi, \lambda_\alpha$: learning rates
τ: soft update parameter for SAC target critic $Q_{\bar{\omega}_i}$
M: maximum number of training episodes
k: growth rate of the number of obstacles

Initialize the network parameters randomly: $\phi, \omega, \bar{\omega}$
Initialize the replay memory $\mathcal{B} \leftarrow \{\}$
for each episode iteration i up to M **do**
  $n_i = \min(\lceil n_0 + k \cdot i \rceil, n_{max})$
  for each iteration j up to n_i **do**
    $obs_j \sim (\mathcal{U}(0, W-1), \mathcal{U}(0, H-1))$
  construct the training map with n_i obstacles
  for each environment step t **do**
    $u_t \sim \pi_\phi(o_t)$ where $u_t = (v_t^{max}, a_t^{max})$
    get d_t via PX4-ROS2 based RL framework
    $\Gamma_t = T(v_t^{max}, a_t^{max}, d_t)$
    move UGV for a fixed period according to Γ_t
    get o_{t+1} via PX4-ROS2 based RL framework
    $\mathcal{B} \leftarrow (o_t, u_t, r_t, o_{t+1})$
  for each gradient step **do**
    $\omega_i \leftarrow \omega_i - \lambda_Q \nabla J_Q(\omega_i)$ for $i = 1, 2$ ▷ by Eq. (3)
    $\phi \leftarrow \phi - \lambda_\pi \nabla J_\pi(\phi)$ ▷ by Eq. (4)
    $\alpha \leftarrow \alpha - \lambda_\alpha \nabla J(\alpha)$ ▷ by Eq. (5)
    $Q_{\bar{\omega}_i} = \tau Q_{\omega_i} + (1-\tau) Q_{\bar{\omega}_i}$ for $i = 1, 2$

Playground and Gazebo are seamlessly integrated within the PX4-ROS2 based RL framework.

The trajectory generation algorithm takes the dynamic states from PX4 as well as v_{max} and a_{max} from the RL agent to generate trajectories. Simultaneously, it transmits the raw observations to the RL environment. The RL environment preprocesses the raw observations it receives and delivers them to the RL agent. The RL agent's actor then extracts v_{max} and a_{max} using the preprocessed observation as input. During this process, the RL environment also provides the RL agent with rewards for observation-action pairs. Using this reward, the RL agent's actor and critic models are updated. Communication between the trajectory generation algorithm and the RL environment is facilitated through the XRCE-DDS middleware.

The trajectory generation algorithm is performed identically in both simulation and the real- world using the same code. This unified data processing helps the RL agent adapt to various real-world scenarios. By minimizing differences between simulation and reality, the algorithm achieves high sim-to-real performance without extra training. Integrating the RL and trajectory generation algorithms enables the agent to adapt to environmental changes, improving stability and efficiency. Thus, the PX4-ROS2 based RL framework effectively handles real-time parameter optimization in dynamic environments and enhances zero-shot sim-to-real transfer performance.

The pseudocode for the proposed SAC training algorithm is shown in Algorithm 1. First, the network parameters and replay buffer are initialized. For each training episode, the curriculum-based random training map is created. For each time step, the trajectory generation algorithm generates per-time step trajectory using the dynamic states d_t as well as v_t^{max} and a_t^{max} extracted through the actor π_ϕ. After the UGV moves along the generated per-time step trajectory for a fixed period, it perceives the dynamically changed surrounding environment in real-time to generate o_{t+1} via the PX4-ROS2 based RL framework. The generated transitions are then stored in the replay buffer for training. For each gradient step, the actor and critic models as well as the temperature parameter model are trained using the objective functions described in Section III-B.

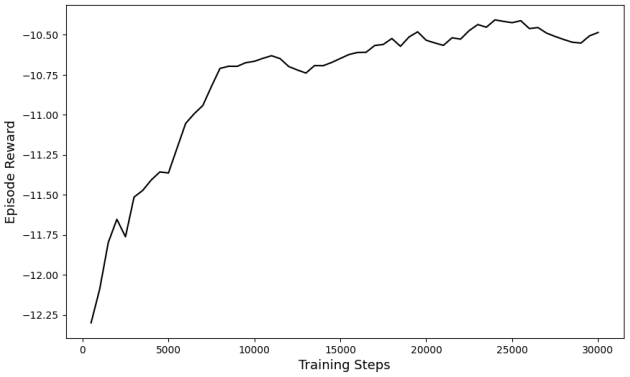


FIGURE 5. Training graph (Episode reward is shown using exponentially weighted moving average smoothing).

V. EXPERIMENTAL RESULTS

A. EXPERIMENTAL SETTING AND TRAINING RESULTS

In our experiment, we apply the EGO-Planner to our UGV trajectory generation. Although EGO-Planner is a trajectory generation algorithm originally designed for UAVs, it has been successfully adapted for use in UGVs in many recent studies and has shown promising results, as mentioned in Section II.

In the original EGO-Planner method, the v_{max} and a_{max} of a UGV are fixed at 1.5 m/s and 6.0 m/s^2, respectively. In contrast, our RL agent dynamically determines the UGV's v_{max} within the range of 1.0 m/s to 4.5 m/s and a_{max} within the range of 6.0 m/s^2 to 10.0 m/s^2. The dynamically determined v_{max} and a_{max} play a significant role in enhancing the trajectory generation process by adapting to the operational constraints of the UGV.

The UGV operates in an unknown environment without prior knowledge beyond its perception range. Due

TABLE 4. The parameter settings of our SAC algorithm.

Parameters	Values
Actor learning rate λ_π	0.001
Critic learning rate λ_Q	0.0001
Entropy learning rate λ_α	0.001
The initial number of obstacles n_0	50
The maximum number of obstacles n_{max}	250
Growth rate of the number of obstacles k	1
Soft update parameter for SAC target critic τ	0.005
Target entropy $\bar{\mathcal{H}}$	0.2
Maximum number of training episodes M	500
Size of training map $W \times H$	$50m \times 50m$
Size of observation $w_o \times h_o$	$8.4m \times 8.4m$

to unpredictable obstacle positions and shapes, the UGV must continuously perceive and adapt in real-time. Initially, RL agent training is conducted in the simulation system using Gazebo and Swarm Playground to learn parameter control performance, and the comparative results are evaluated. We then conduct real-world experiments by applying both the EGO-Planner method and the proposed method to the UGV. The detailed components of the UGV used in these experiments are shown in Fig. 6. The primary objective is to evaluate the impact of the proposed method on velocity changes by comparing the velocity, yaw speed, and path of the UGV while avoiding obstacles. In simulations, small obstacles are evenly distributed, whereas in the real-world, larger obstacles are sparsely located.

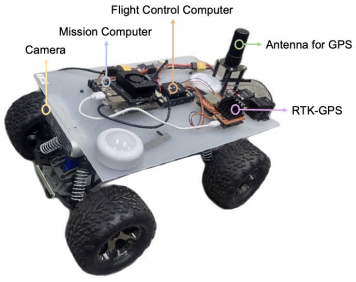
The RL agent's actor and critic models are trained using the SAC algorithm, which is implemented based on the *skrl* library [47]. The parameters used for training are detailed in Table 4. After the training process is completed, only the actor model trained by the SAC algorithm is utilized during inference to determine the v_{max} and a_{max}.

Fig. 5 shows the episode reward over 30,000 training steps of the proposed RL algorithm. To emphasize the trend, an exponentially weighted moving average smoothing technique was used, and the resulting smoothed reward is depicted with a solid black line. The consistent increase in the smoothed episode reward demonstrates the effective learning and convergence of the algorithm.

B. SIMULATION EXPERIMENTS

Fig. 7 presents the comparative results of the proposed method and the original EGO-Planner method after training in the simulation. The UGV moves from start to destination following paths generated by the algorithms, with the red path representing the proposed method's trajectory and the orange path representing the EGO-Planner's trajectory. The proposed method results in a higher average speed compared to the EGO-Planner method in each test environment, which reduces UGV travel time by approximately 28 on average. The figure show that the proposed method achieves faster travel times and also improves the overall smoothness of the path. This improvement in path smoothness contributes to more efficient and stable navigation for the UGV.

Fig. 8 shows the changes in velocity and yaw speed between 8 and 16 seconds for each test. To represent the changes in yaw speed, a Gaussian filter with a sigma value of 50 is used. As shown in the figure, while the velocity in the EGO-Planner method change within a very narrow range, the proposed method produces dynamically changing velocities. Moreover, in the proposed method, the yaw speed also changes significantly in synchronization with these varying velocities.



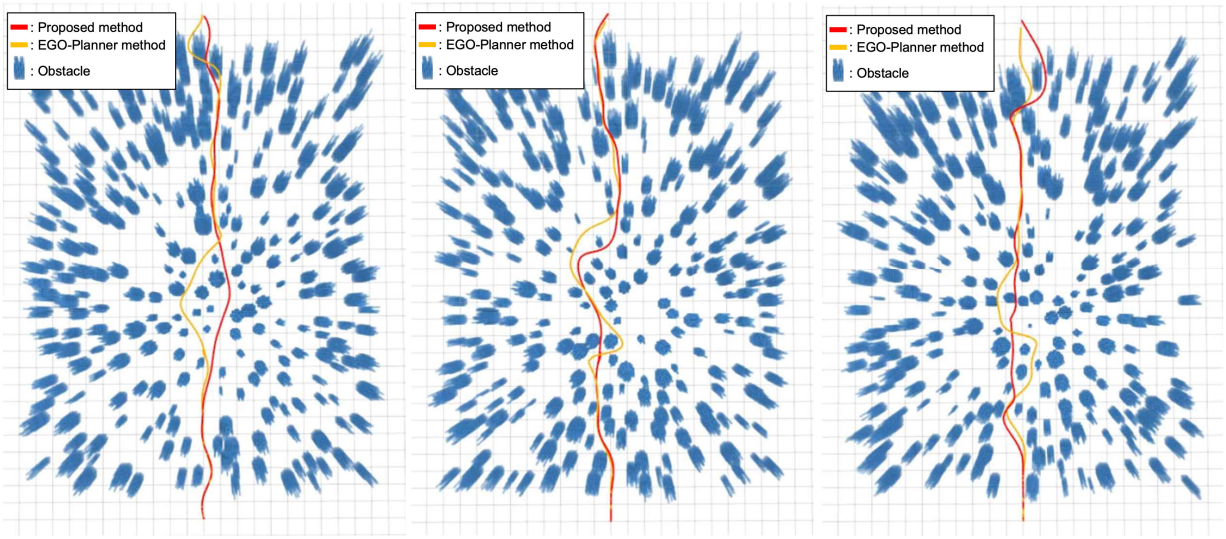
Hardware Components of UGV	Model
Antenna for GPS	HX-CH6601A
RTK-GPS	Piksi Multi
Flight Control Computer	PX4 autopilot
Mission Computer	NVIDIA Xavier NX
Antenna for DDS	CN9 2dbi
Camera	RealSense D455

FIGURE 6. The UGV used in real-world experiments.

More specifically, significant changes in yaw speed indicate that the UGV is entering areas near obstacles. In these situations, the proposed method demonstrates a pattern where the UGV avoids accelerating beyond a certain speed before approaching these areas and then accelerates while passing through them. This indicates that our RL agent is well trained to consider potential unseen obstacles and reduces velocity to prevent collisions before approaching areas near obstacles. Once the UGV enters such areas, it accelerates to quickly pass through the hazardous zone, thereby ensuring both the safety of the UGV and reduced travel time. These instances are highlighted in gray in the figure.

C. REAL-WORLD TEST

The comparative experiments in the real-world is conducted through zero-shot sim-to-real transfer without additional training or tuning. The real-world tests are conducted three times in each of the three environments, and the comparative experimental results are shown in Fig. 9. In the first environment, the UGV navigates 14 meters forward while successfully avoiding a single detected obstacle. The second environment includes two obstacles, where the UGV navigates 15 meters forward while successfully avoiding the two detected obstacles. The third environment features two obstacles that are widely spaced apart, where the UGV



(a). Test environment I (b). Test environment II (c). Test environment III

	Proposed method		EGO-Planner method	
	Average velocity (m/s)	Travel time (s)	Average velocity (m/s)	Travel time (s)
Test environment I	1.79	19.54	1.21	26.26
Test environment II	1.86	18.44	1.25	25.91
Test environment III	1.70	18.86	1.15	26.82

FIGURE 7. Comparative inference results in the simulation experiments.

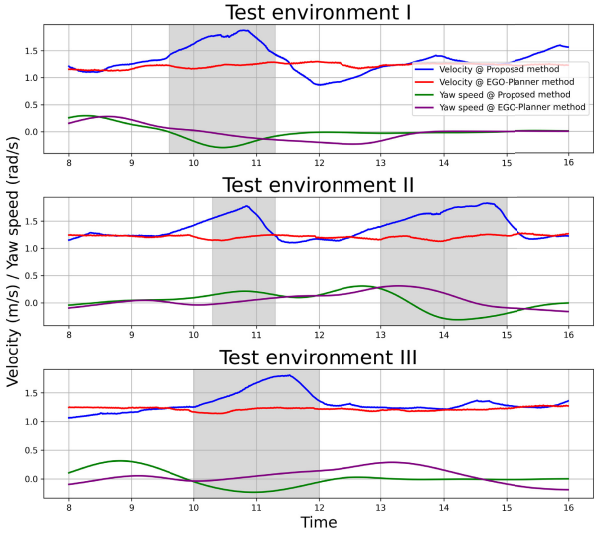


FIGURE 8. Comparison of velocity and yaw speed in the simulation experiments.

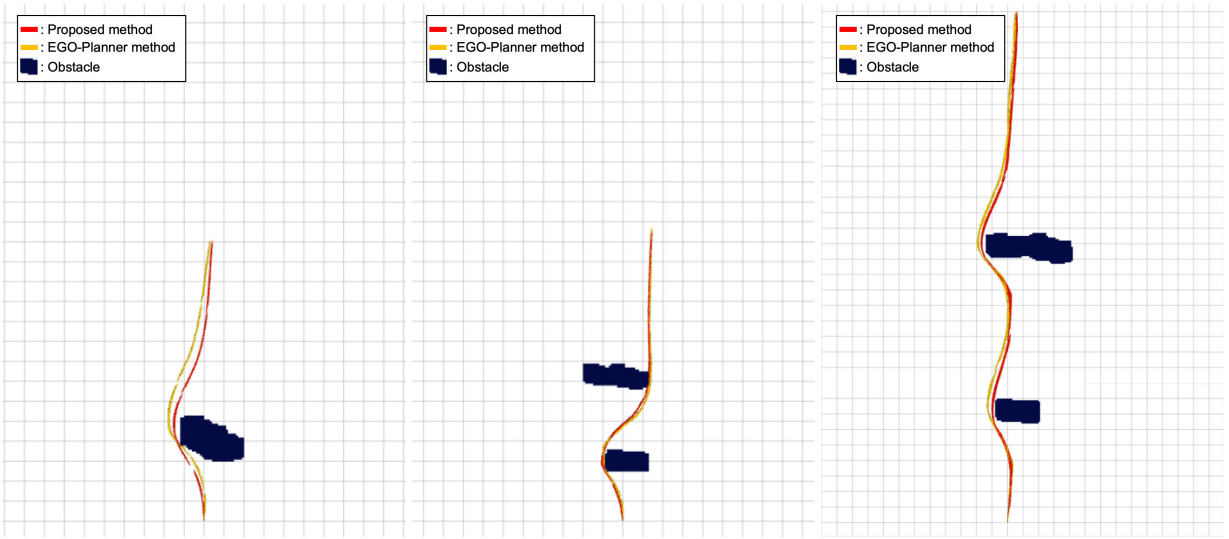
navigates 33 meters forward while successfully avoiding the two detected obstacles. For the three environments, the figure shows the detected obstacles, the actual path of the UGV, the average velocity, and the travel time for both the EGO-Planner method and the proposed method. The experimental results demonstrate improvements in average speed and reductions in travel time, which are consistent with the results of the simulation experiments.

Fig. 10 presents a comparison of velocity and yaw speed across the three environments. The figure shows the mean values and standard deviations for both the proposed method and the EGO-Planner method. Similar to the results presented in Fig. 8, Fig. 10 shows that the proposed method follows a similar velocity change pattern in the real-world experiments. The UGV avoids accelerating beyond a certain speed before approaching areas near obstacles and then accelerates while passing through them. These instances are highlighted in gray in the figure. In real-world experiments, the RL agent also considers potential unseen obstacles and reduce velocity before approaching them. Once the UGV enters these areas, it accelerates to quickly pass through, ensuring both safety and reduced travel time.

D. ZERO-SHOT SIM-TO-REAL TRANSFER VALIDATION

Finally, we conduct experiments in both simulation and real-world environments with identical obstacle configurations. Through these experiments, we compare the velocity and yaw speed to verify if their change patterns are similar in both the simulation and real-world environments. Similar to Fig. 9 (c), we organize the simulation environment with two widely spaced obstacles, and conduct the experiments accordingly. The simulation experiments are conducted three times and the results are compared with the real-world data given by Fig. 9 and Fig. 10.

Fig. 11 visualizes the changes in velocity and yaw speed in each experiment with their means and standard



(a). Single obstacle (b). Two obstacles (c). Two widely spaced obstacles

	Proposed method		EGO-Planner method	
	Average velocity (m/s)	Travel time (s)	Average velocity (m/s)	Travel time (s)
Single obstacle	1.21 ± 0.04	10.73 ± 0.34	1.06 ± 0.05	11.83 ± 0.84
Two obstacles	1.26 ± 0.01	11.50 ± 0.04	1.12 ± 0.01	12.16 ± 0.04
Two widely spaced obstacles	1.38 ± 0.06	25.47 ± 0.55	1.14 ± 0.01	26.85 ± 0.52

FIGURE 9. Comparative inference results in the real-world experiments.

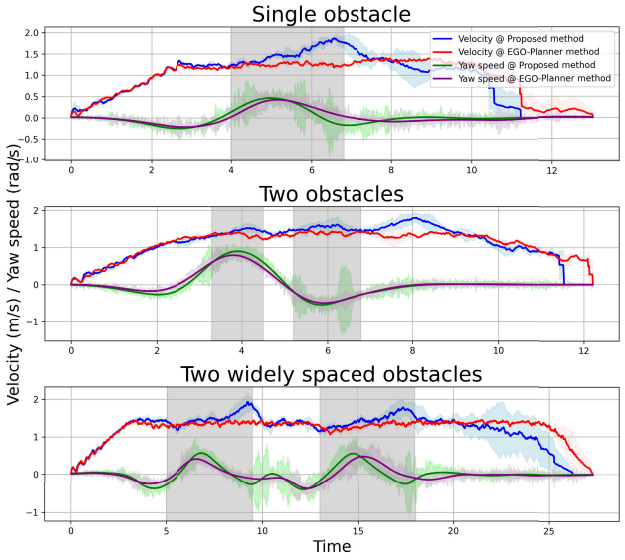


FIGURE 10. Comparison of velocity and yaw speed in real-world experiments.

deviations. The results demonstrate that the RL agent operates consistently in both simulation and real-world environments. This consistent velocity and yaw speed pattern indicates that the RL agent reliably operates despite the differences between simulation and real-world environments. Through these results, we confirm that the proposed method successfully implements zero-shot sim-to-real transfer over the PX4-ROS2 based RL framework.

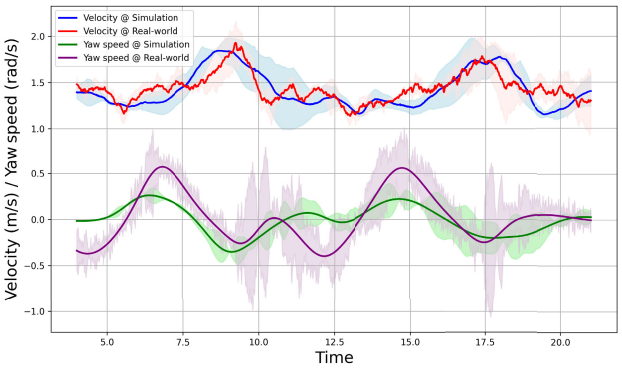


FIGURE 11. Comparison of velocity and yaw speed in simulation and real-world environments with two widely spaced obstacles.

VI. CONCLUSION

For UGV trajectory generation, traditional fixed parameter settings fail to adapt to environmental changes, resulting in inefficient and unsafe trajectories. To overcome this limitation, we propose a new RL-based method for real-time optimization of the maximum velocity (v_{max}) and maximum acceleration (a_{max}) to generate a UGV trajectory in dynamic environments. Moreover, to ensure the stable sim-to-real transfer of the proposed method, we introduce a new PX4-ROS2 based RL framework. This framework unifies the

operation between simulation and real-world environments and allows the RL models trained in simulation to perform effectively in the real-world. As a result, the zero-shot sim-to-real transfer is achievable without additional training or tuning in the real-world. The proposed RL-based parameter control method significantly enhances performance in terms of efficient and safe trajectory generation in real-world UGV autonomous navigation applications. The algorithm's ability to adapt in real time to various environmental changes improves the safety and efficiency of autonomous UGV operations in rapidly changing environments.

Most UGV operating environments are unknown and dynamic. In such environments, our research improves UGV performance by controlling the parameters of the trajectory generation algorithm. Consequently, the proposed approach is widely applicable to various UGV navigation tasks, such as reconnaissance and search and rescue operations. Notably, since our study includes real-world validation through zero-shot sim-to-real transfer, it offers the possibility of immediate implementation. In our future research, we will propose a new RL-based method to control not only the maximum speed and maximum acceleration but also more critical parameters of the trajectory generation algorithm.

REFERENCES

[1] J. Tordesillas, B. T. Lopez, M. Everett, and J. P. How, "FASTER: Fast and safe trajectory planner for navigation in unknown environments," *IEEE Trans. Robot.*, vol. 38, no. 2, pp. 922–938, Apr. 2022.

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

[3] J. Tordesillas and J. P. How, "MADER: Trajectory planner in multiagent and dynamic environments," *IEEE Trans. Robot.*, vol. 38, no. 1, pp. 463–476, Feb. 2022.

[4] Y. Wang, J. O'Keeffe, Q. Qian, and D. Boyle, "KinoJGM: A framework for efficient and accurate quadrotor trajectory generation and tracking in dynamic environments," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2022, pp. 11036–11043.

[5] S. Zhang, X. Liu, B. Yan, J. Bi, and X. Han, "A real-time look-ahead trajectory planning methodology for multi small line segments path," *Chin. J. Mech. Eng.*, vol. 36, no. 1, p. 59, Apr. 2023.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[7] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: A survey," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2020, pp. 737–744.

[8] E. Salvato, G. Fenu, E. Medvet, and F. A. Pellegrino, "Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning," *IEEE Access*, vol. 9, pp. 153171–153187, 2021.

[9] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 12619–12629.

[10] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 8973–8979.

[11] Y. Du, O. Watkins, T. Darrell, P. Abbeel, and D. Pathak, "Auto-tuned sim-to-real transfer," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 1290–1296.

[12] K. Arndt, M. Hazara, A. Ghadirzadeh, and V. Kyrki, "Meta reinforcement learning for sim-to-real domain adaptation," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 2725–2731.

[13] D. Liu, Y. Chen, and Z. Wu, "Digital twin (DT)-CycleGAN: Enabling zero-shot sim-to-real transfer of visual grasping models," *IEEE Robot. Autom. Lett.*, vol. 8, no. 5, pp. 2421–2428, May 2023.

[14] I. Higgins, A. Pal, A. Rusu, L. Matthey, C. Burgess, Al. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner, "DARLA: Improving zero-shot transfer in reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1480–1490.

[15] X. Zhou, X. Yu, Y. Zhang, Y. Luo, and X. Peng, "Trajectory planning and tracking strategy applied to an unmanned ground vehicle in the presence of obstacles," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 4, pp. 1575–1589, Oct. 2021.

[16] H. Ren, S. Chen, L. Yang, and Y. Zhao, "Optimal path planning and speed control integration strategy for UGVs in static and dynamic environments," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 10619–10629, Oct. 2020.

[17] B. Brito, B. Floor, L. Ferranti, and J. Alonso-Mora, "Model predictive contouring control for collision avoidance in unstructured dynamic environments," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 4459–4466, Oct. 2019.

[18] Y. Chen, H. Dong, and Z. Li, "Asynchronous spatial allocation protocol for trajectory planning of heterogeneous multi-agent systems," *arXiv:2309.07431*, 2023.

[19] H. Liu, W. Dong, Z. Zhang, C. Wang, R. Li, and Y. Gao, "Optimization-based local planner for a nonholonomic autonomous mobile robot in semi-structured environments," *Robot. Auto. Syst.*, vol. 171, Aug. 2024, Art. no. 104565.

[20] Z. Li, R. Mao, N. Chen, C. Xu, F. Gao, and Y. Cao, "ColAG: A collaborative air-ground framework for perception-limited UGVs' navigation," 2023, *arXiv:2310.13324*.

[21] X. Zhou, X. Wen, Z. Wang, Y. Gao, H. Li, Q. Wang, T. Yang, H. Lu, Y. Cao, C. Xu, and F. Gao, "Swarm of micro flying robots in the wild," *Sci. Robot.*, vol. 7, no. 66, May 2022, Art. no. eabm5954.

[22] C. Wang, D. Deng, L. Xu, and W. Wang, "Resource scheduling based on deep reinforcement learning in UAV assisted emergency communication networks," *IEEE Trans. Commun.*, vol. 70, no. 6, pp. 3834–3848, Jun. 2022.

[23] R. Ding, F. Gao, and X. S. Shen, "3D UAV trajectory design and frequency band allocation for energy-efficient and fair communication: A deep reinforcement learning approach," *IEEE Trans. Wireless Commun.*, vol. 19, no. 12, pp. 7796–7809, Dec. 2020.

[24] G. Karafotias, A. E. Eiben, and M. Hoogendoorn, "Generic parameter control with reinforcement learning," in *Proc. Annu. Conf. Genetic Evol. Comput.*, Jul. 2014, pp. 1319–1326.

[25] R. Chen, B. Yang, S. Li, and S. Wang, "A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem," *Comput. Ind. Eng.*, vol. 149, Nov. 2020, Art. no. 106778.

[26] Y. Chen, X. Lin, T. Khan, and M. Mozaffari, "Efficient drone mobility support using reinforcement learning," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, May 2020, pp. 1–6.

[27] W. J. Yun, S. Jung, J. Kim, and J.-H. Kim, "Distributed deep reinforcement learning for autonomous aerial eVTOL mobility in drone taxi applications," *ICT Exp.*, vol. 7, no. 1, pp. 1–4, Mar. 2021.

[28] U. Challita, W. Saad, and C. Bettstetter, "Deep reinforcement learning for interference-aware path planning of cellular-connected UAVs," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–7.

[29] S. Josef and A. Degani, "Deep reinforcement learning for safe local planning of a ground vehicle in unknown rough terrain," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 6748–6755, Oct. 2020.

[30] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2021, pp. 1205–1212.

[31] A. Romero, Y. Song, and D. Scaramuzza, "Actor-critic model predictive control," 2023, *arXiv:2306.09852*.

[32] E. Valassakis, Z. Ding, and E. Johns, "Crossing the gap: A deep dive into zero-shot sim-to-real transfer for dynamics," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 5372–5379.

[33] T. Bi, C. Sferrazza, and R. D'Andrea, "Zero-shot sim-to-real transfer of tactile control policies for aggressive swing-up manipulation," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 5761–5768, Jul. 2021.

[34] V. Dasagi, R. Lee, S. Mou, J. Bruce, N. Sünderhauf, and J. Leitner, "Sim-to-real transfer of robot learning with variable length inputs," 2018, *arXiv:1809.07480*.

[35] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, 2014.

[36] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.
[37] L. Meier, D. Honegger, and M. Pollefeys, "PX4: A node-based multi-threaded open source robotics framework for deeply embedded platforms," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2015, pp. 6235–6240.
[38] R. Raveendran, S. Ariram, A. Tikanmäki, and J. Röning, "Development of task-oriented ROS-based autonomous UGV with 3D object detection," in *Proc. IEEE Int. Conf. Real-time Comput. Robot. (RCAR)*, Sep. 2020, pp. 427–432.
[39] S. Ray, R. Bhowal, P. Patel, and K. A. Panaiyappan, "An overview of the design and development of a 6 DOF remotely operated vehicle for underwater structural inspection," in *Proc. Int. Conf. Commun., Control Inf. Sci. (ICCISc)*, vol. 1, Jun. 2021, pp. 1–6.
[40] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Sci. Robot.*, vol. 7, no. 66, May 2022, Art. no. eabm6074. https://www.science.org/doi/abs/10.1126/scirobotics.abm6074
[41] G. Pardo-Castellote, "OMG data-distribution service: Architectural overview," in *Proc. 23rd Int. Conf. Distrib. Comput. Syst. Workshops, Proceedings.*, 2003, pp. 200–206.
[42] *DDS for EXtremely Resource-Constrained Environments (XRCE-DDS)*, Object Manage. Group, Needham, MA, USA, 2019.
[43] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *Proc. Int. Conf. Eng. Technol. (ICET)*, Aug. 2017, pp. 1–6.
[44] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," 2014, *arXiv:1406.1078*.
[45] T. Ni, B. Eysenbach, and R. Salakhutdinov, "Recurrent model-free RL can be a strong baseline for many POMDPs," 2021, *arXiv:2110.05038*.
[46] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, vol. 3, Jun. 2004, pp. 2149–2154.
[47] A. Serrano-Muoz, D. Chrysostomou, S. Bgh, and N. Arana-Arexolaleiba, "SKRL: Modular and flexible library for reinforcement learning," *J. Mach. Learn. Res.*, vol. 24, no. 254, pp. 1–9, 2023.

CHANG-HUN JI received the B.S. and M.S. degrees from Korea University of Technology and Education, Cheonan, South Korea, in 2020 and 2023, respectively, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering. His research primarily focuses on advanced topics in artificial intelligence, with a particular emphasis on reinforcement learning. Additionally, he is exploring the development and optimization of trajectory generation algorithms for unmanned vehicles, aiming to enhance their autonomous navigation capabilities in dynamic environments.

GYEONGHUN LIM received the B.S. degree from Korea University of Technology and Education, Cheonan, South Korea, in 2023. He is currently pursuing the M.S. degree with the Department of Intelligent Systems and Robotics, Chungbuk National University, Cheongju, South Korea. His research primarily focuses on cooperative driving using unmanned ground vehicles, aiming to enhance the efficiency and safety of autonomous systems through advanced coordination and communication techniques among multiple vehicles. Additionally, he is exploring innovative algorithms and strategies to improve the robustness of swarm intelligence in dynamic and complex environments.

YOUN-HEE HAN (Member, IEEE) received the B.S. degree in mathematics and the M.S. and Ph.D. degrees in computer science and engineering from Korea University, Seoul, South Korea, in 1996, 1998, and 2002, respectively. From 2002 to 2006, he was a Senior Researcher with the Next Generation Network Group, Samsung Advanced Institute of Technology. Since 2006, he has been a Professor with the School of Computer Science and Engineering, Korea University of Technology and Education, Cheonan, South Korea. From September 2013 to January 2015, he was a Visiting Professor with the Department of Computer Science, State University of New York (SUNY) at Albany. Since 2002, his activities have focused on mobility management, media independent handover, and cross-layer optimization for efficient mobility support. He is very interested in artificial intelligence technology, especially reinforcement learning, and he has been in charge of many research projects regarding improving the performance of reinforcement learning algorithms for various fields, such as intelligent networking on 5G and 6G, the Internet of Things (IoT), economics, and financial engineering. He has made several contributions to IETF and IEEE standardization. He actively participated in the IEEE 802.21 Working Group. He is the author of IETF RFC 5181, RFC 5270, and RFC 7864. He has published approximately 270 research articles on the theory and application of mobile computing and has led 45 patents regarding the information and communication technology domain. His current research interests include the theory and application of computer networks, including protocol design and mathematical analysis, mobile sensor/actuator networks, social network analysis, machine learning, deep learning, and reinforcement learning.

SUNGTAE MOON received the B.S. degree from Chonnam National University, Gwangju, South Korea, in 2005, the M.S. degree from Gwangju Institute of Science and Technology (GIST), Gwangju, in 2007, and the Ph.D. degree in aeronautical engineering from KAIST, in 2021. From 2007 to 2010, he was with the Agency for Defense Development (ADD), where he developed a mission computer for aircraft. From 2011 to 2012, he was with the National Security Research Institute (NSRI) and worked in the area of security in embedded systems. From 2012 to 2022, he was a Senior Researcher with the Artificial Intelligence Research Division, Korea Aerospace Research Institute (KARI). From 2022 to 2023, he was with Korea University of Technology and Education. Since 2023, he has been with Chungbuk National University (CBNU), Cheongju, South Korea, where he is currently an Assistant Professor with the Department of Intelligent System and Robotics. His current research interests include swarming flight systems, navigation algorithms, and object detection based on deep learning.