Integrated Learning-based Framework for Autonomous Quadrotor UAV Landing on a Collaborative Moving UGV

Chang Wang[1]†, Jiaqing Wang[2]†, Changyun Wei*[2], Kailei Qi[2], Ze Ji[3], Zhaowei Ma[1] and Mingjin Xu[4]

***Abstract*—Autonomous unmanned aerial vehicle (UAV) landing on a moving unmanned ground vehicle (UGV) remains a challenge as it is difficult for the UAV to track the real-time state of the UGV and adjust its landing policy accordingly. In this paper, we propose a learning framework for a quadrotor UAV to land on a moving UGV without knowing its motion dynamics. Specifically, the learning framework consists of two main systems: a Landing Vision System (LVS) using deep learning and a Landing Control System (LCS) using deep reinforcement learning. The LVS enables the UAV to recognize and localize the UGV in real-time for estimating the relative position and velocity between them. Besides, the location of the UGV is tracked in the field of view of the UAV using consecutive images which alleviates the problem of tracking failure. We propose an algorithm, named Memory Consolidated TD3 (MCTD3), for generating optimal policies to enable precise tracking and landing control of the UAV. In addition, we propose an adaptive COACH (ACOACH) algorithm that allows human intervention in the action space of the UAV to speed up the training process. We demonstrate the effectiveness of the proposed method in both simulation and real-world experiments.**

***Index Terms*—UAV landing, autonomous landing, convolutional neural networks, YOLO, deep reinforcement learning, bio-inspired system.**

I. Introduction

Unmanned aerial vehicles (UAVs) and unmanned ground vehicles (UGVs) have been widely used in a variety of real-world applications, including agriculture [1], logistics [2], reconnaissance [3], and search-and-rescue [4]. Coordination between UAVs and UGVs is often more practical for solving complex, interdependent tasks than using them individually. For instance, when a quadrotor UAV hovers at a certain height, it can share a global view of the terrain with a UGV for efficient path planning. On the other hand, the UGV can provide the UAV with a battery charging facility to extend the flight time.

[1]†Chang Wang, Zhaowei Ma are with the College of Intelligence Science and Technology, National University of Defense Technology, Changsha, 410073, China

[2]†Jiaqing Wang, Changyun Wei and Kailei Qi are with the College of Mechanical and Electrical Engineering, Hohai University, Changzhou, 213022, China

[3]Ze Ji is with the School of Engineering, Cardiff University, Cardiff CF24 3AA, United Kingdom

[4]Mingjin Xu is with the College of Naval Architecture and Ocean Engineering, Naval University of Engineering, Wuhan,430033, China

This work was supported in part by the National Natural Science Foundation of China under Grant 61906203 and 62303481, and in part by the University Independent Scientific Research Project under Grant 2022505070.

*Corresponding author: Changyun Wei. (e-mail: c.wei@hhu.edu.cn)

†These authors contributed equally to this work.

Autonomous landing is essential for a UAV to collaborate with a UGV [5], [6]. If reliable GPS coordinates of the UAV and the landing destination were provided, the landing task could be solved as a control problem. This can be achieved through the use of fuzzy control [7], Model Predictive Control (MPC) [8], PD (Proportional, Derivative) control [9] , PID (Proportional, Integral, Derivative) control [10], and other techniques. Unfortunately, these methods may fail in GPS-denied environments, where the UAV's or the destination's coordinates are inaccurate [11]. To tackle this issue, the UAV can apply vision-based control methods to track and locate the destination, resulting in the use of vision combined with reinforcement learning to address the landing task [11]–[14].

In our previous work [5], [15], we integrated the Deep Deterministic Policy Gradient (DDPG) [16] reinforcement learning algorithm with the traditional PID controller to achieve autonomous tuning of PID parameters. However, the UAV would occasionally lose track of the UGV in its field of view, revealing the need to provide the UAV with reliable target tracking and localization [17] along with landing control.

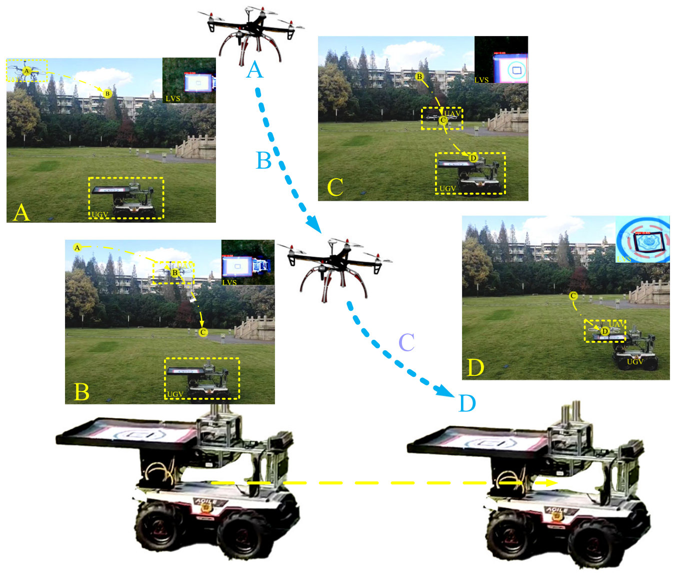


Fig. 1. Tracking and landing process of our proposed autonomous landing system.

In this paper, we propose an integrated learning framework consisting of two systems: the onboard Landing Vision System (LVS) for UGV detection and the Landing Control System

(LCS) for UAV motion control, as illustrated in Fig. 1. The main contributions are summarized as follows:

- The integrated learning framework enables reliable UAV tracking of the UGV while learning the optimal landing policy.
- We design a lightweight pipeline to achieve rapid detection and localization of the moving UGV, which can provide accurate relative positions and velocity estimations between the UAV and UGV.
- We propose an efficient deep reinforcement learning algorithm MCTD3 for learning the landing policy, and we propose an interactive learning algorithm ACOACH to incorporate human experience for more efficient learning.

The rest of the paper is organized as follows: Section II presents the related preliminaries of deep reinforcement learning and the kinematic model of the landing task. We illustrate our system framework in Section III. Our proposed LVS and LCS are elaborately described in Section IV and V. Section VI discusses the experimental results. Finally, we conclude the paper in Section VII.

II. Preliminaries

A. Deep Reinforcement Learning and TD3

Deep Reinforcement Learning (DRL) [18] combines the advantages of deep neural networks and reinforcement learning (RL) to learn control policies for high-dimensional or continuous state spaces. The Q-value $Q^{\pi}(s_t, a_t)$ is defined as the expected cumulative future discounted rewards for an action a_t taken at the time step t. The goal of RL is to find the optimal policy that maximizes the expected return. In a deterministic policy μ, the Q-value is defined by the Bellman expectation condition:

$$Q^{\mu}(s_t, a_t) = \mathbb{E}\left[r_t + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))\right], \tag{1}$$

where s_{t+1} is the state at the time step $t+1$ and $\pi(a_t \mid s_t)$ represents the policy that maps from the state space to the action space. Neural networks are used to calculate the Q-values for each state-action pair. A function approximator is designed to minimize the error through the trained and optimized parameter θ^*, as follows:

$$\theta^* = \arg\min_{\theta} \mathbb{E}\left[(\delta_t - Q_{\theta}(s_t, a_t))^2\right], \tag{2}$$

where $\delta_t = r_t + \gamma \max_a Q_{\theta}(s_{t+1}, a)$ is defined as the Temporal Difference error (TD-errors), essentially the difference between the predicted and actual values of the reward.

Although Q-value-based approaches are useful in searching for the optimal policy μ_{ϕ}, parameterized by ϕ, a more efficient approach is to use the policy gradient method to maximize the accumulated reward $J\left(\mu_{\phi}\right) = \mathbb{E}_{\mu_{\phi}}\left[\sum_t R(s_t, a_t)\right]$. The gradients are used to update ϕ and is calculated as [19].

Actor-critic methods combine both value-based and policy-gradient methods [20]. When the agent interacts with the environment at the time step t, the state s_t is observed, and the agent receives an action command a_t from the policy $\pi(a_t \mid s_t; \theta)$ accordingly.

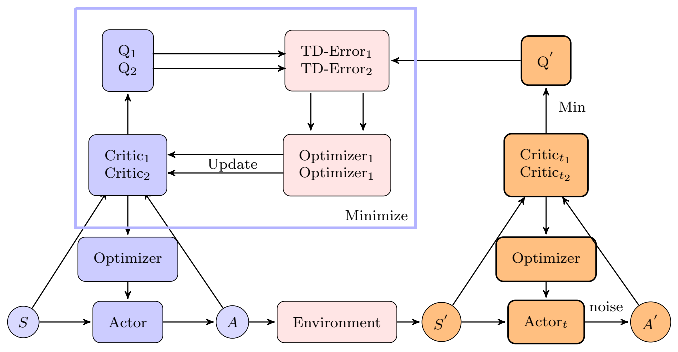


Fig. 2. Framework of TD3

Twin delayed deep deterministic policy gradient (TD3) represents one of the most successful applications of the actor-critic-based reinforcement learning paradigm, as shown in Fig. 2. One of the known issues with value-based reinforcement learning methods is that the Q-function tends to overestimate the Q-value, leading to suboptimal policies due to function-approximation errors. TD3 addresses this problem by constraining the overestimated Q-value by taking the minimum value between a pair of critic networks $Q_{\theta_i}(s, a)_{i=1,2}$, where $\theta_{i(i=1,2)}$ are the parameters of the critic networks. Furthermore, Gaussian noises is added to the actions generated from the actor network to fully explore the environment. The Bellman equation is utilized to calculate a target value $y(r, s')$. The smaller item of the two outputs $\left(Q_{\theta'_1}, Q_{\theta'_2}\right)$ from the target critic networks is fed into the Bellman equation to avoid overestimating the Q -value as:

$$y(r, s') = r + \gamma \min_{i=1,2} Q_{\theta'_i}\left(s', \pi_{\phi}(s') + \varepsilon\right). \tag{3}$$

We then update the critic networks as:

$$\theta_i \leftarrow \operatorname{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2 \tag{4}$$

Meanwhile, the target actor network is not updated consecutively to reduce the overestimation problem. In TD3, however, the parameter ϕ will be updated by the deterministic policy gradient after certain training iterations, in the author's case, e.g., updated the target critic networks two times [21]. The equation for the policy update is described as follows:

$$\nabla_{\phi} J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)\Big|_{a=\pi_{\phi}(s)} \nabla_{\phi} \pi_{\phi}(s). \tag{5}$$

Finally, a hyper-parameter $\tau (0 < \tau < 1)$ is introduced for the soft update, which deals with reduce the problem of overfitting during the training. The target networks are updated as follows:

$$\begin{aligned} \theta'_i &\leftarrow \tau\theta_i + (1-\tau)\theta'_i \\ \phi' &\leftarrow \tau\phi + (1-\tau)\phi'. \end{aligned} \tag{6}$$

B. Coordinate System

The Coordinate System used in our study consists of several sub-systems, including the Pixel Coordinate System (PCS), the Camera Coordinate System, the Rack Coordinate System, the World Coordinate System (WCS), and the Target Coordinate Systems (TCS), as shown in Fig. 3. These sub-systems are essential for understanding the position and orientation of the UAV. In our design, the on-board camera is fixed to the UAV Rack. Therefore, we combine the Camera Coordinate System with Rack Coordinate System as Camera & Rack Coordinate System (CCS). We also examined a quadrotor UAV with a rack in the X-configuration. The rack provides upward forces through four mini-motors installed perpendicular to the rack and rotation in the O_CZ_C negative direction. The pipline for the UAV camera's transformation of the PCS into the WCS is shown Fig. 4.

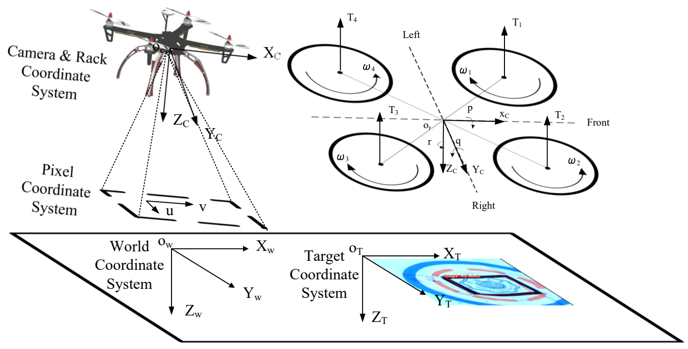


Fig. 3. Definition of the coordinate system and forces acting on the UAV.

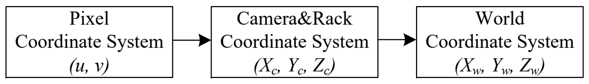

Pixel Coordinate System (u, v)
Camera&Rack Coordinate System (X_c, Y_c, Z_c)
World Coordinate System (X_w, Y_w, Z_w)

Fig. 4. Conversion process from the PCS to the WCS.

III. Framework

The problem of UAV landing can be represented as a Markov Decision Process (MDP). When the UAV detects the UGV, it can estimate their relative position and velocity, which allows for the calculation of the state vector. We define the MDP for the landing task as $\{S, A, P, R, \gamma\}$. The current state s_t leads to the next state s_{t+1} once the UAV perceives the current state s_t and executes the current action a_t, as depicted in Figure 5.

1) state:
   The state vector s_t at the time step t is defined as $(\Delta d_{x_t}, \Delta d_{y_t}, \Delta d_{z_t}, \Delta v_{x_t}, \Delta v_{y_t}, D_t) \in S$, where S represents a six-dimensional continuous state space. $(\Delta d_{x_t}, \Delta d_{y_t}, \Delta d_{z_t})$ denotes the relative distances of the UAV to the UGV in the WCS at the time step t, and $(\Delta v_{x_t}, \Delta v_{y_t})$ represents the relative velocity between the UAV and the Landing Platform (LP) in the x and y axes. D_t serves as a warning variable indicating whether the UAV is approaching the boundary of the LP.
2) action:
   The action space A is defined as a three-dimensional continuous space. The action a is represented as $a = (v_x, v_y, v_z) \in A$, where v_x, v_y, and v_z denote the reference velocities sent to the UAV's velocity controller along the x, y, and z axes in the WCS.
3) policy:
   Solving the MDP involves determining a policy $\pi_\theta \in \Pi : S \times A \rightarrow [0, 1]$ within a finite time horizon T, where π_θ is parameterized by θ. The policy P represents the conditional probability $p(s' \mid s, a)$. It is defined as a mapping from the state-action space $(S \times A)$ to a probability distribution over the next state space S, denoted as $P : S \times A \rightarrow P(S)$.

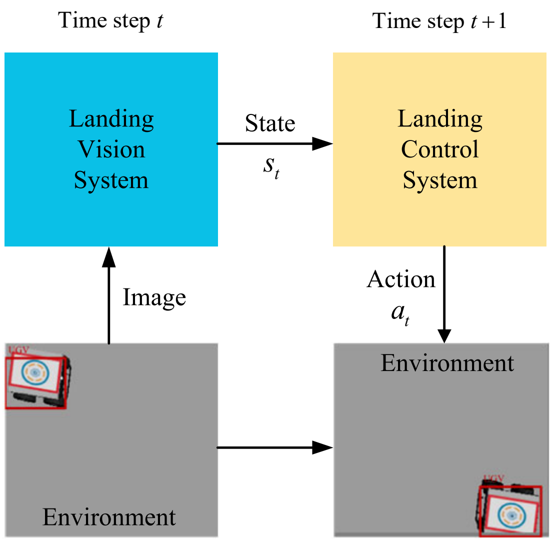


Fig. 5. The state transition for UAV autonomous landing.

4) reward function:
   Formulating an effective landing policy P necessitates the creation of a well-defined reward function R.

The realization of each reward r_t at discrete time steps comprises three components: tracking rewards, r_{dist} and r_{velo}, and landing guidance reward, r_{land}.

- Distance reward r_{dist}: The design of r_{dist} is to encourage the agent to minimize the relative distance at each step. The relative distance between the UAV and the target is calculated as $d_t = \sqrt{\Delta d_{x_t}^2 + \Delta d_{y_t}^2}$. The tracking process is deemed successful if d_t meets the distance constraint d_{in} at the time step t. On the other hand, the tracking process is considered failed if d_t exceeds the limited distance d_{lim}. d_{out} represents the maximum safe distance for tracking. Fig. 6 illustrates the distance setup for item r_{dist}. Additionally, we record an approach step $n_{approach}$ to encourage continuous tracking. $n_{approach}$ increments by 1 when d_t is within the range of d_{in} and resets to 1 when the limit is exceeded. $n_{approach}$ has a maximum of 5 steps to prevent overflow of data. Moreover, an indicator variable D_t is incorporated into the state space. D_t is set to -1 if the distance between the UAV and the target is larger than d_{in}. The indicator variable D_t acts as a warning signal, alerting the UAV when it approaches the boundary of the LP.

$$r_{dist} = \begin{cases} (d_{in} - d_t) \cdot n_{approach}, & \text{if } d_t \le d_{in} \\ D_t, & \text{if } d_{in} < d_t \le d_{out} \\ D_t - d_t, & \text{if } d_{out} < d_t \le d_{lim} \end{cases} \tag{7}$$

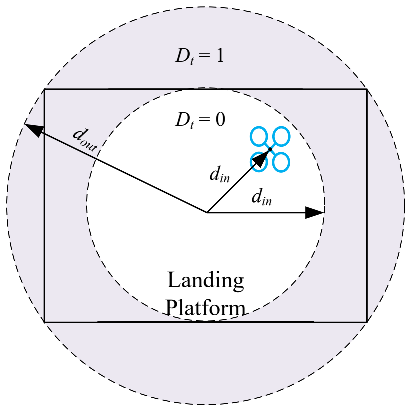


Fig. 6. Reward setup.

- Velocity reward r_{velo}: In the r_{velo} design, $\Delta v_t = \sqrt{\Delta v_{x_t}^2 + \Delta v_{y_t}^2}$ represents the UAV's velocity relative to the LP. We employ the velocity delimiter Δv_{in} to prevent sudden acceleration and ensure stability. If the velocity difference Δv_t adheres to the Δv_{in}, then the tracking process is deemed successful at the the time step t. However, agent recives a penalty of $-\Delta v_t$ when the velocity difference exceeds Δv_{in}.

$$r_{velo} = \begin{cases} (\Delta v_{in} - \Delta v_t) \cdot n_{approach}, & \text{if } \Delta v_t \le \Delta v_{in} \\ -\Delta v_t, & \text{if } \Delta v_{in} \le \Delta v_t \end{cases} \quad (8)$$

- Landing guidance reward r_{land}: The design of r_{land} aims to incentivize the agent to land successfully on the LP. If the relative distance between UAV and UGV is within the range of d_{in} in the termination steps, a positive reward of $200 - d_t$ is given; otherwise, a penalty of -200 given. The agent receives more rewards if it lands closer to the center of the LP.

$$r_{land} = \begin{cases} 200 d_{in} - 200 dt & \text{if } \Delta z_t < h \text{ and } d_t \le d_{in} \\ -200 & \text{else} \end{cases} \quad (9)$$

As a result, the reward function is as follows:

$$r_t = \omega_1 \cdot r_{dist} + \omega_2 \cdot r_{velo} + \omega_3 \cdot r_{land}, \quad (10)$$

where ω_1, ω_2, and ω_3 are the weight coefficients assigned to each reward component.

IV. Landing Vision System

The Landing Vision System (LVS) aims to provide the UAV with the capability to recognize, locate and track the target UGV. Specifically, we consider the relative position and velocity between the UAV and UGV to construct a state vector that enables the UAV to learn the optimal landing policy. YOLO5 is used for target detection. The coordinates of the bounding box are then used to extract D_t, Δd_{x_t}, and Δd_{y_t}. The UAV's altitude Δz_t is determined using the Depth Estimator (DE) module. Finally, the relative velocities, Δv_{x_t} and Δv_{y_t}, are computed using a velocity estimation algorithm, as shown in Fig. 7.

A. Depth Estimator

We develop the Depth Estimator (DE) using a Convolutional Neural Network (CNN), which estimates the distance between an UAV and a UGV. To reduce noises and increase computational efficiency, we first crop the images to obtain our Region of Interest (ROI) before sending it to the vision system pipeline. Cropping reduces the amount of data and improves the computational efficiency of our landing approach. The LVS continually estimates the relative position and velocity of the UGV with respect to the UAV during the autonomous landing process, therefore we select bounding box of the UGV and a ten-pixel region around it as the Region of Interest (ROI). We feed this ROI into the DE pipeline. We construct a feature extraction architecture in DE using two convolutional layers and two subsampling layers, respectively. Specifically, the ten channels in the first convolution layer are fed by the input layer's single channel, and each of the ten kernels (each with a size of 5×5) represents a feature between two corresponding channels. The ReLU [22] (Rectified Linear Unit) is used to introduce non-linearity to the convolution layer. The last fully-connected layer's output provides the direct distance estimation.

We pre-trained our YOLOv5 model using a dataset comprising images of the Landing Platform (LP) taken at various altitudes in the simulation. In testing, we validated the accuracy of the weights used in YOLOv5. We optimized the network in DE training using SGD (Stochastic Gradient Descent) with a momentum of 0.9 and a learning rate of 10^{-3}. We used the Huber loss function to update the DE network. We initialized all layer weights from a normal Gaussian distribution with a mean of zero and a standard deviation of 0.01, while the biases were initialized to 0.1. Unlike previous YOLOv5 regression training, we did not prepare any dataset or labels for the DE training. Instead, we commenced DE training and the LCS training simultaneously using Gazebo. During each training step t, we fed a 640×480 image into the DE network, along with real-time altitude information read from Gazebo to serve as the label over the LVS training process. The training ended after the DE's prediction error was lower than 0.2 for 15 times continuously.

B. Velocity estimator

To estimate the velocity of a moving object captured by a camera, we propose a methodology that involves converting pixel coordinates of the target in two frames to the WCS. Suppose we have images of a moving object captured at time steps t_1 and t_2, with pixel coordinates P_1 and P_2 from equation (**??**). We derive corresponding camera coordinate positions P_{C_1} and P_{C_2} of the target in the PCS as:

$$P_{C_1} = \Delta z_{t_1} K^{-1} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix}, \ P_{C_2} = \Delta z_{t_2} K^{-1} \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix}. \quad (11)$$

We then transform P_{C_1} and P_{C_2} from the CCS to the WCS. The distance between the UGV's two positions in the WCS is calculated as $d = \|P_{W2} - P_{W1}\|$, where $\|\cdot\|$ denotes the Euclidean

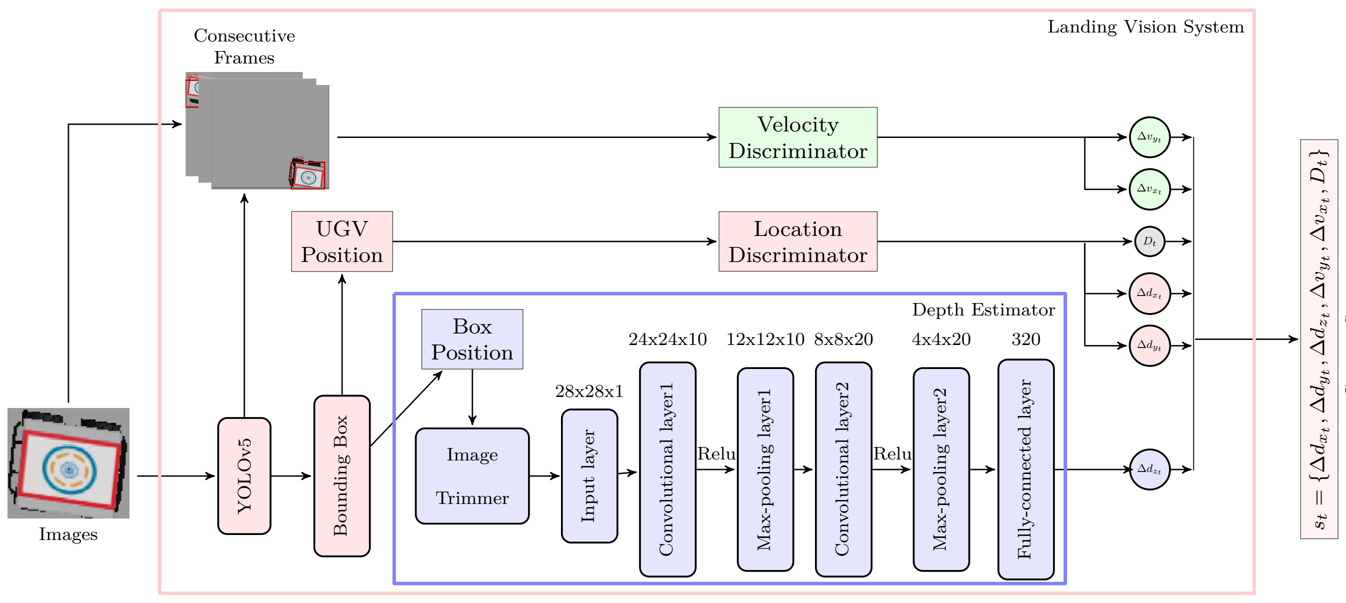


Fig. 7. Pipeline of the LVS. YOLOv5 is used with a CNN network to calculate the UAV's relative position to the UGV.

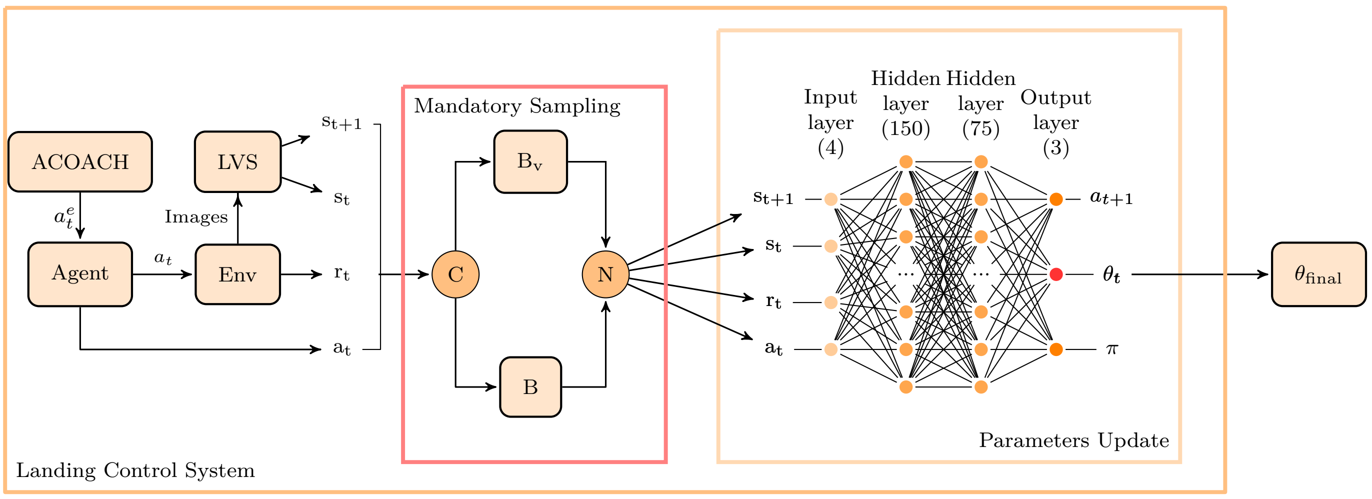


Fig. 8. Pipeline of the Landing Control System.

norm. Finally, we can estimate the velocity v_t of the moving platform in the WCS using the following equation:

$$v_t = \frac{d}{\Delta t} \cdot \frac{1}{\sqrt{f^2 + 1}} \cdot S, \tag{12}$$

where Δt is the time interval between the two frames, S is the scale of the image in pixels per meter, and f is the focal length of the camera lens in millimeters. The velocity components (v_{x_t} and v_{y_t}) of the unmanned ground vehicle (UGV) can be estimated using the Sine and Cosine Theorems.

V. Landing Control System

Deep reinforcement learning for robotic control presents two challenges. First, if the agent was trapped in failed trials and used those samples for training, it would prematurely terminate the learning process [23]. Second, it is challenging for the agent to discover an optimal policy in reward-sparse environments [24]. In other words, the agent needs to experience sufficient successful trials and explore efficiently to collect high-quality training samples. To address these challenges, we propose the Landing Control System (LCS) that consists of the Memory Consolidated TD3 (MCTD3) and the Adaptive COACH (ACOACH), as shown in Fig. 8.

A. *Memory Consolidated TD3*

The proposed Memory Consolidated TD3 (MCTD3) algorithm has two parts: the Mandatory Sampling and the TD-error Discriminator.

1) Mandatory Sampling: This approach draws inspiration from the Partitioned Buffer Replay method [25] and the Prioritized Experience Replay method [11].

The MS method utilizes an additional replay buffer B_v, which is specifically designed to record exceptional transitions. After the completion of a landing trial, the terminal transition is copied into B_v for five times. Besides, we use the Prioritized Experience Replay (PER) method to draw samples from the replay buffer B. PER samples according to the probabilities based on the TD-errors from the critic network updates. Compared to random sampling, PER can help the agent learn more effectively in an environment with sparse reward. Subsequently, N_{batch} samples are drawn from both buffers to update all critic networks. Actor networks are updated every N_{policy} iterations. Finally, the parameters of the critic networks and the actor network are updated together.

2) TD-error Discriminator: It is used to advice the direction of the update process base on the work of Hasselt et al. [26]. During training, taking an action a_t that results in a positive change in the state-value function has the potential to enhance the policy by maximizing the discounted future reward. To encourage the RL agent to select significant transitions with direct impacts on the policy, we adopt the following approach:

$$\text{If} \quad \delta_t > 0 \quad : \quad \text{increase}_{\,t}\left(\pi_t\left(s_t, a_t\right)\right) \tag{13}$$

where increase$_t(\pi(s_t, a_t))$ is the increase in the probability of selecting action a_t in state s_t.

In summary, the learning process of MCTD3 begins with random initialization of the network weights, clearing both buffers B and B_v. Next, the agent selects an action with exploration noise N_t and interacts with the environment, and recives the resulting transitions. Subsequently, N_{batch} samples are drawn from buffers B and B_v to update all networks based on the TD errors. Finally, the parameters of the target critic networks $\theta^{Q'}_{i(i=1,2)}$ and the parameter of the target actor network $\theta^{\mu'}$ are updated. The Memory Consolidated TD3 (MCTD3) algorithm is summarized in Algorithm 1.

Algorithm 1 Memory Consolidated TD3

```
1:  $\theta^{Q'}_i \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$;   ▷ Initialize target network with random weights.
2:  $B \leftarrow \emptyset$, $B_v \leftarrow \emptyset$;   ▷ Initiate memory buffers $B$ and $B_v$.
3:  for $t = 1$ to $T$ do:
4:      Select and execute $a_t + N_t$, Obtain $s_{t+1}$ with $r_t$;
5:      Store $(s_t, a_t, r_t, s_{t+1})$ in $B$ with maximal priority $p_i$ ;
6:      if the terminal condition is satisfied then:
7:          Store the termination transition $(s_t, a_t, r_t, s_{t+1})$ in $B_v$ for $N_{B_v}$ times ;
8:          for $i = 1$ to $N_{update}$ do:
9:              Draw $\frac{N_{batch}}{2}$ samples from $B$ with priority $p_i$ and another $\frac{N_{batch}}{2}$ from $B_v$ randomly;
10:             if $(\delta_t > 0)$ then:
11:                 Update $\theta^{Q'}_{i(i=1,2)}$ using equation (3), (4), (6);
12:                 Update $\theta^{\mu'}$ using equation (5), (6);
13:             end if;
14:         end for;
15:     end if;
16:     Update priority $p_i$, start a new episode;
17: end for.
```

B. Adaptive COACH

The COACH (COrrective Advice Communicated by Humans) method [27] provides online feedback from human during the RL process. In this framework, the signal $A_e : S \rightarrow \mathbb{R}$ represents the human feedback with respect to the state space. By utilizing information from A_e, COACH computes an adaptive learning rate $\alpha(s_t)$ to facilitate learning of expert experiences by the agent. A higher learning rate allows for larger corrections by the human supervisor, while a lower learning rate enables finer adjustments. The adaptive learning rate is computed as:

$$\alpha(s_t) = |A_e(s_t)| + bias \tag{14}$$

where *bias* is the default value of the learning rate.

In this paper, we propose the Adaptive COACH (ACOACH) method by introducing the Action Smoother to enhance the performance of the COACH method. In traditional imitation learning, sparse signals from expert demonstrations can hinder the generalizability of policies [28]. Specifically, the RL agent learns that performing actions in multiple dimensions simultaneously is more effective than taking one-dimensional actions at each time step. Therefore, we propose a regularization strategy that smoothes the received actions as follows:

$$\begin{aligned} \tilde{a}^e &\leftarrow \eta({a_t}^e + \frac{\xi}{2}({a_{t-1}}^e + {a_{t-2}}^e)) \\ \eta &\sim \frac{\Delta z}{H}, \quad \xi \sim N\left(\mu, \sigma^2\right), \end{aligned} \tag{15}$$

where ξ is drawn from a Gaussian distribution and superimposes the previous actions with the take-off height H of the UAV, enabling the agent to efficiently learn action combinations in the later training stages.

Whenever ACOACH receives an action signal ${a_t}^e$ provided by the human expert at the time step t, it replaces the action a_t generated by MCTD3 and executes the smoothed action ${\tilde{a}_t}^e$. ${s_t}^e$ and ${r_t}^e$ correspond to the states and rewards obtained from the environment through expert instruction ${\tilde{a}_t}^e$, respectively. ACOACH then utilizes these transitions to update the target MCTD3 networks whenever the agent receives the current reward ${r_t}^e$, the current state s^e_{t+1}, and the previous state s^e_t based on expert instructions. Algorithm 2 outlines the procedure of ACOACH.

Algorithm 2 ACOACH

```
Input: $\theta^{Q'}_i$, $\theta^{\mu'}$;   ▷ Network parameters from Algorithm 1.
Output: $\theta^{Q'}_i$, $\theta^{\mu'}$   ▷ New parameters for Algorithm 1.
1:  while Algorithm 1 is running do:
2:      Record current state ${s^e}_t$;
3:      Take action $a_t$ accoding to the current policy $P(s_t)$.
4:      ${a_t}^e \leftarrow$ gethumanCorrectiveAdvice()
5:      if ${a_t}^e$ != 0 then:
6:          Get the smoothed action ${\tilde{a}_t}^e$ using equation (15);
7:          Take action ${\tilde{a}_t}^e$ , obtain $s^e_{t+1}$ and $r^e_t$ from the environment;
8:          Update $\theta^{Q'}_{i(i=1,2)}$, $\theta^{\mu'}$ using equation (6), (14).
9:      end if;
10: end while.
```

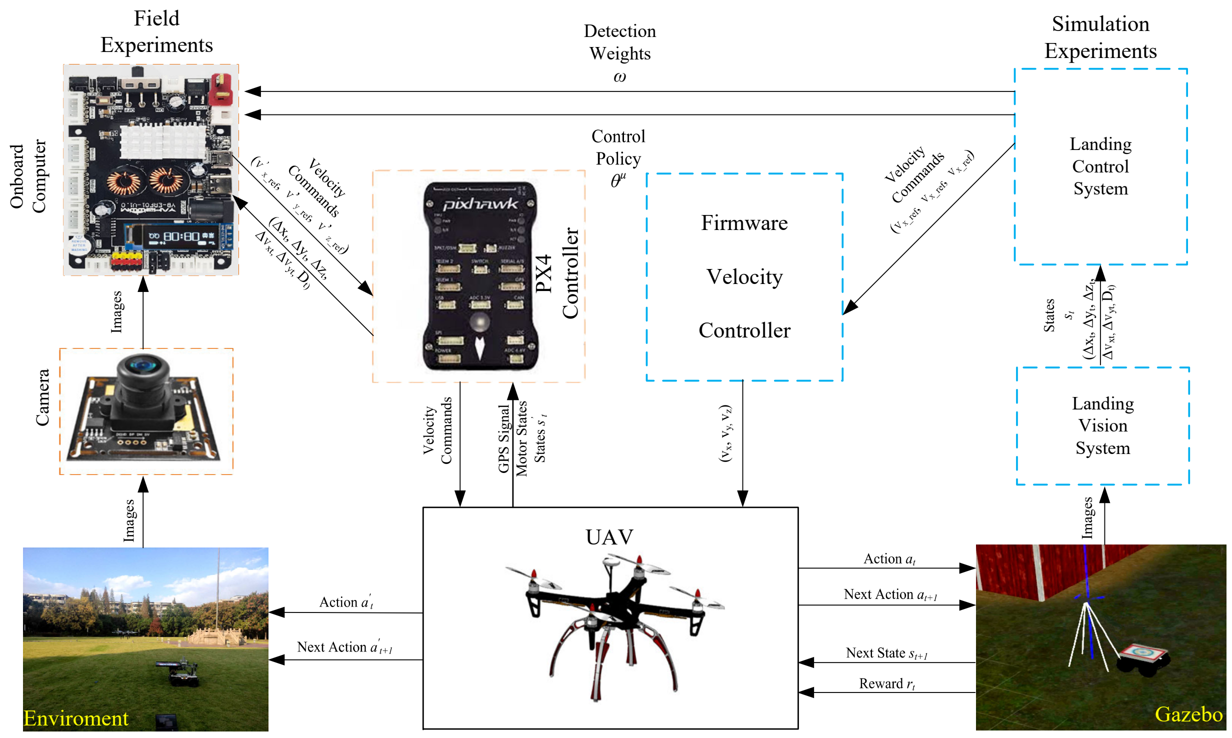


Fig. 9. Components and data stream in the simulation and real-world experiments.

VI. Experiments and Results

A. Experimental Settings

The experiments consisted of two parts, i.e. simulation and real-world as shown in Fig. 9. The training of all the controllers was conducted in the Gazebo simulation environment. MAVLink was used to connect the PX4 Autopilot flight controller[*] computer. During our simulated experiments, we utilized a modified Iris UAV equipped with a camera mounted beneath it, perpendicular to the UAV frame. Importantly, the connection between the UAV and the on-board camera remained fixed for these experiments. Furthermore, we modified the top plate of a Husky UGV[†] model to mount a detection marker.

We used the Adam optimizer [29] for the actor network and two critic networks during the simulated training phase. All hidden layers in the neural networks used the Leaky Relu [30]. The actor-critic network architecture is presented in Table I. Throughout the training and testing stages, we simulated high-wind conditions for landing by adding continuous Gaussian noise to the UAV maneuver commands. The supplementary action noises were limited to a magnitude of 0.4 m/s and lasted no more than 2 seconds. The training parameters are presented in Table II. For our real-world experiments, we transferred the algorithms and learned policies directly from the simulation to the on-board firmware of our custom-built S450 UAV platform.

TABLE I
Settings Of Networks.

	Input	Network Dimensions	Output	Operator
	s_t	6× 150	FC1	Leaky Relu
Actor	FC1	150× 75	FC2	Leaky Relu
	FC2	75× 3	a_t	tanh
	$s_t + a_t$	9× 200	FC1	Leaky Relu
Critics	FC1	200× 150	FC2	Leaky Relu
	FC2	150× 1	Q	/

TABLE II
Parameters description

Parameters	Definition	Value
T	Training iterations	400
N	Replay buffer size of B	1.6×10^5
N_v	Replay buffer size of B_v	1.6×10^3
N_{B_v}	Copies for termial transitions	5
N_{update}	Update iterations	100
N_{batch}	Batch size	16
N_{policy}	Policy update delay	2
N_t	Exploration noise	0.2
γ	Discount factor	5×10^{-3}
α	Learning rate of actor network	10^{-4}
α_1	Learning rate of critic network 1	10^{-4}
α_2	Learning rate of critic network 2	10^{-3}
τ	Learning rate of the soft update	5×10^{-3}

B. Training in Simulation

At the start of the training, the UAV took off to 3.5 m above the UGV from the same side, and the UGV kept moving forward. In the first ten training episodes, an operator guided the UAV to land using ACOACH. After an episode ended,

[*]https://px4.io/
[†]https://robots.ros.org/husky/

the positions of both the UGV and UAV were reset for the next episode. Three factors were taken into account when determining the trial termination condition. First, it is time-consuming to prepare the UAV for re-takeoff once the PX4 activates the landing mode. Secondly, we aimed to prevent the UAV from colliding with any parts of the UGV during training for safety reasons. Finally, we assumed the UAV is capable of landing successfully at a height of 20 *cm* above the safe-landing zone of the LP by utilizing PX4's landing mode.

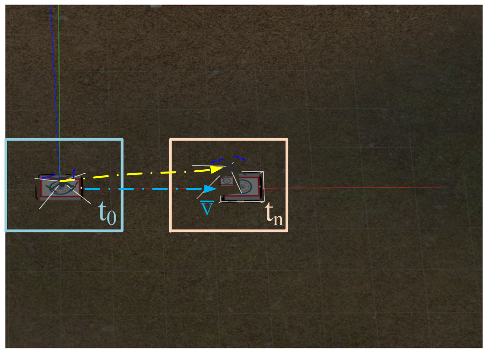
The training process utilized a mini-batch size N_{batch} of 32 with 10 iterations whenever the UAV landed, while a Gaussian noise was added to the action values with a standard deviation of 0.2 during the exploration. We trained three algorithms (TD3, TD3+ACOACH, and LCS) under the same conditions in two landing scenarios. Each algorithm was used for 10 trials, with 400 episodes per trial. We averaged the accumulated reward along with the success records obtained in the training trials, and the learning curves are presented in Fig. 10 and Fig. 11. In the training process of LCS and TD3+ACOACH, an expert employs ACOACH (Algorithm 2) to deliver action instructions a_t^e to the agent through a keyboard in the Gazebo simulation environment for the initial ten training episodes.

(1) Training Scenario A: UGV movement along a straight trajectory with an uniform speed.

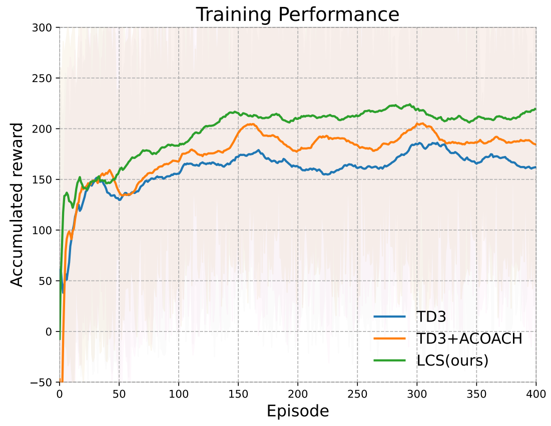
Training trials were initially conducted with a consistent UGV speed of 0.2 m/s under scenario A, as depicted in Fig. 10(a). The learning curves of our proposed models LVS and TD3+ACOACH, along with a TD3 baseline, are presented in Fig. 10(b). Additionally, Fig. 10(c) demonstrates a significant increase in the success rate curves of all three algorithms between episodes 0 and 70, indicating the effectiveness of our training design in developing an efficient landing approach. Both the TD3+ACOACH algorithm and the LCS, which utilizes the ACOACH, outperformed TD3. This suggests that the ACOACH significantly enhances stability and convergence speed in less complex scenarios. However, not much difference was observed among the three methods in Fig. 10(b) and 10(c). We attribute this lack of distinction to the insufficient difficulty of the experimental settings. Therefore, further training was conducted under Scenario B.

(2) Training Scenario B: UGV movement along a curved trajectory with sudden accelerations. In order to further validate the effectiveness of the proposed landing method, we conducted more challenging trainings under Training Scenario B. A faster and more variable speed of the UGV can increase the complexity of automated landing task, as depicted in Fig. 11(a). We set the velocity of the UGV to vary randomly between 0.2 m/s and 0.8 m/s every 5 seconds during the training trials. Random sudden velocity changes ranging from -0.2 m^2/s to 0.2 m^2/s were applied to the UGV's motion every 5 seconds during the training trials.

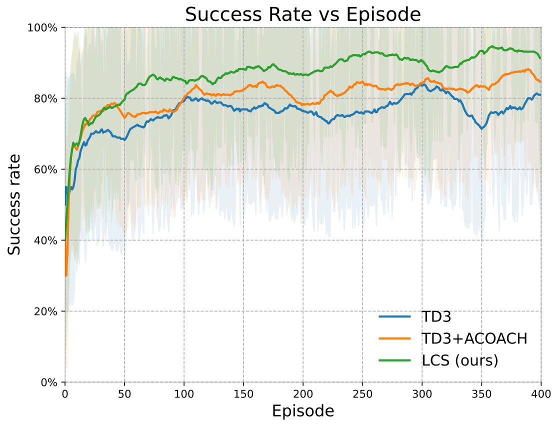
We present the learning curves of our proposed models LVS, TD3+ACOACH, and the baselines in Fig 11(b). It is noteworthy that TD3 failed to achieve satisfactory performance within 400 training episodes, as indicated by its success rate climbing in Fig. 11(c). This poor performance can be attributed to the replay buffer being overwhelmed with negative rewards, which reduces the chances of updating the Critics and leads to



(a) Training diagram of Scenario A in the Gazebo simulator.



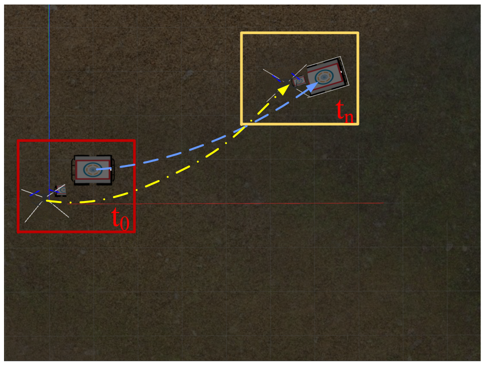
(b) Learning curve for accumulated reward.



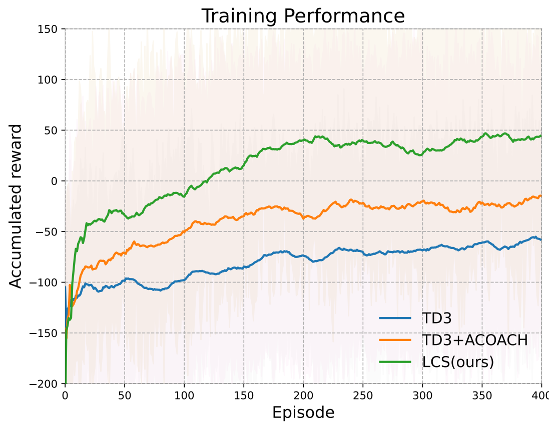
(c) Learning curve for success rate.

Fig. 10. Comparison of averaged accumulated reward and averaged success rate for three algorithms (TD3, TD3+ACOACH, LCS) in Scenario A.

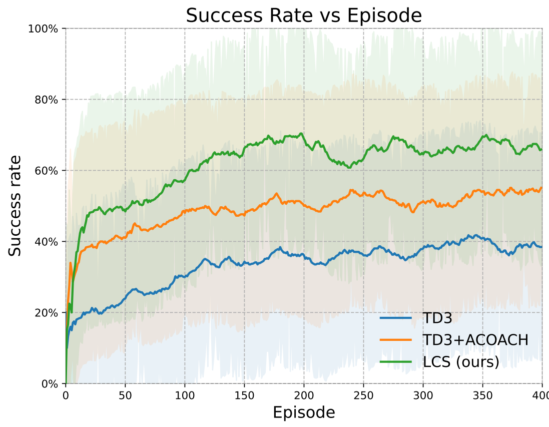
a lower learning efficiency. The ACOACH module increases the probability of discovering the optimal landing policy by exploring higher-rewarded state and action spaces. The proposed LCS algorithm demonstrates better performance than both TD3 and TD3+ACOACH. Additionally, LCS maintains a higher reward than other agents starting from the 50th episode. This can be attributed to the utilization of the MCTD3, which allows the agent to refine its original update strategy and improve learning efficiency and convergence rate. Finally, our results indicate that the learning curves of TD3+ACOACH and LCS exhibit more stability and rapidity compared to TD3,



(a) Training diagram of Scenario B in the Gazebo simulator.



(b) Learning curve for accumulated reward.



(c) Learning curve for success rate.

Fig. 11. Comparison of averaged accumulated reward and averaged success rate for three algorithms (TD3, TD3+ACOACH, LCS) in Scenario B.

further highlighting the benefits of the ACOACH method.

C. Testing in Simulation

We evaluated the landing performance of our learned policies using three metrics: landing success rate, time cost, and landing precision. To test the effectiveness of our policies under various UGV speeds, we performed experiments using four constant speed values: $v_{ugv} = 0\ m/s$ (static), $0.2\ m/s$, $0.5\ m/s$, and $1\ m/s$. Since Training Scenario B incorporates Training Scenario A, we implemented the landing strategies trained for each controller in Training Scenario B. We repeated the experiments for 100 trials for each of the three algorithms (TD3, TD3+ACOACH, LCS) to assess their performance.

1) Success rate and time cost: The success rates and average time costs are presented in Fig. 12. We averaged the success rates and time costs over all testing trials, see Table III.

TABLE III
AVERAGED SUCCESS RATE AND TIME COST IN TESTING

Algorithm	Success Rate	Time
TD3	58.75%	$6.741s$
TD3+ACOACH	68.50%	$7.198s$
LCS	**83.75%**	$7.489s$

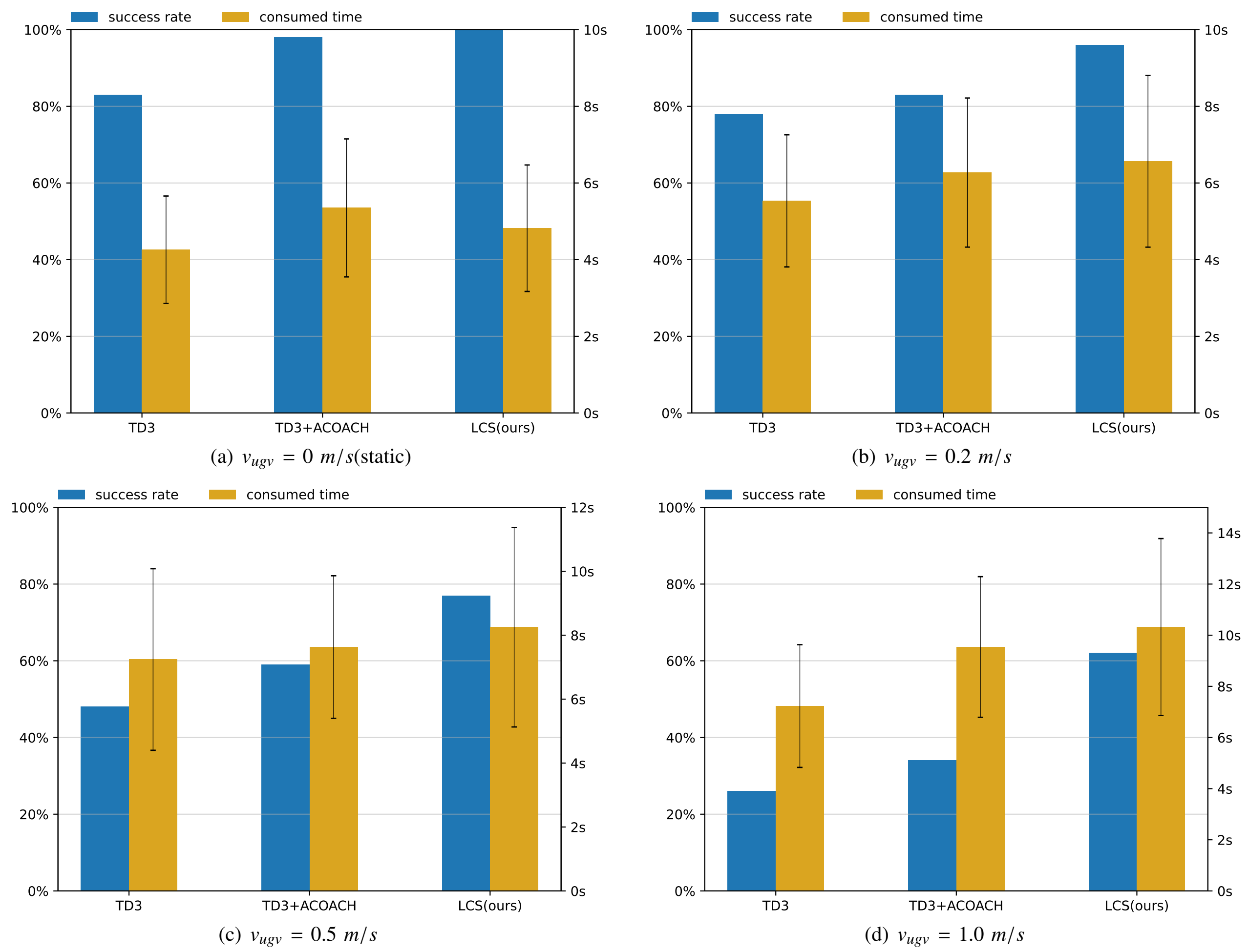
Based on our analysis of Fig. 12 and Table III, we can conclude that LCS outperformed the other two algorithms. In the static UGV scenario, LCS achieved an almost 100% success landing rate, while TD3 had a significantly lower success rate. As the UGV speed v_{ugv} increased, the success rate of UAV landings gradually decreased for all three methods. At UGV velocity of $0.5\ m/s$, LCS had a success rate of 77%, TD3+ACOACH had a success rate of approximately 60%, and TD3 had a success rate of 49%. When the UGV speed was raised to $1\ m/s$, the success rate of LCS dropped to 62%. This result supports the hypothesis that increasing UGV velocities result in a more complicated landing task.

2) Landing precision: Under the $v_{ugv} = 0.5\ m/s$ condition, we collected and compared the landing points of 100 trials for each of the three algorithms. A safe and precise landing requires proximity to the center of the UGV. We designated 75% of the $0.8\ m/s \times 0.6\ m/s$ marker as the safe-landing zone, depicted by the black rectangle in Fig. 13. The distribution statistics of the landing points are presented in Table IV.

Fig. 13 displays the landing accuracy of the three algorithms under the condition of $v_{ugv} = 0.5\ m/s$. Each circle corresponds to nearly 95.44% (2σ criteria) of the landing points generated by the corresponding algorithm, presuming a normal distribution of the landing points. LCS and TD3+ACOACH demonstrated significantly better performance than TD3. Specifically, LCS and TD3+ACOACH were able to identify superior landing policies that led to safe and precise landings close to the center of the UGV, while TD3 learned sub-optimal policies that resulted in potentially dangerous landings near the edges of the nestling marker.

3) Tracking and Landing Trajectories: Fig. 14 illustrates the landing trajectories of the three algorithms (TD3, TD3+ACOACH, LCS) under the condition of $v_{ugv} = 0.5\ m/s$. Both successful and failed landings are shown in the figure. The moving trajectories of the UGV are depicted by red lines, while the final positions of the LP are represented by black rectangles. The landing trajectories of the UAV are denoted by blue lines. This visualization enables us to better understand the effectiveness of the policies learned by the different methods.

Fig. 14(d) and Fig. 14(a) shows that the landing points of TD3 were predominantly located near the edges of the UGV, which can also be seen in Fig. 13. Additionally, the



(a) $v_{ugv} = 0\ m/s$(static) (b) $v_{ugv} = 0.2\ m/s$ (c) $v_{ugv} = 0.5\ m/s$ (d) $v_{ugv} = 1.0\ m/s$

Fig. 12. Testing results of landing success rates and cost time under different UGV speed conditions using three algorithms (TD3, TD3+ACOACH, LCS) using policy trained from Scenario B.

TABLE IV
STATISTICS OF THE LANDING POINTS

Algorithm	x-axis average distance(cm)	y-axis average distance(cm)	Total average distance(cm)	$\sigma(cm)$	Safe landing count
TD3	26.1 ± 29.3	15.8 ± 13.7	30.5	32.3	43
TD3+ACOACH	17.6 ± 19.8	17.6 ± 14.7	24.8	24.5	62
LCS	**6.5** ± **12.1**	**8.2** ± **14.1**	**10.5**	**18.6**	**84**

UAV took longer periods to hover and showed less elegant diving behavior near the UGV during low-altitude approach, as compared to other successful landing trajectories. However, TD3+ACOACH demonstrated a more deliberate approach before vertically diving towards the UGV, as displayed in Fig. 14(e), while the trajectory of LCS appeared more coherent, as seen in Fig. 14(f). To further demostrate the tracking and landing effect of the LCS, we recorded a representative motion trajectory, as shown in Fig. 16. The less optimal landing policies of TD3 may explain the lower landing success rate as compared to TD3+ACOACH and LCS. Contrarily, the UAV's most common failure was its inability to locate the UGV, where LCS produced the most reasonable landing trajectory among the algorithms, as shown in Fig. 14(c).

We collected 10 successful tracking and landing trajectories for each controller on the position and velocity of the UAV relative to the Landing Platform (LP) in both the x and y axes while examining landing accuracies, and shown in Fig. 15. Each algorithm underwent 10 successful landing trajectories, and the average distance and velocity errors in the x and y for these trajectories are listed in Table V. The LCS algorithm exhibited the highest level of performance with a smoother operation, higher stability, and no significant fluctuations compared to those of TD3 and TD3+ACOACH. The landing trajectories generated by the LCS algorithm exhibited the lowest average distance error of 27.3 cm and the lowest average velocity error of 21.3 cm. Both Fig. 15(a) and Fig. 15(b) demonstrates LCS's superior performance in preventing tracking loss, thereby improving motion stability and safety throughout the landing process. This is a significant improvement over our previous work [5] and the work of Zhao et al. [32]. The results presented in the figures demonstrate

TABLE V
STATISTICS OF THE LANDING TRAJECTORIES

Algorithm	Average distance error(*cm*)				Average velocity error(*cm*)			
	x-axis	*y*-axis	σ	Total	*x*-axis	*y*-axis	σ	Total
TD3	64.3 ± 48.6	38.1 ± 35.4	60.1	74.7	40.2 ± 58.6	25.8 ± 31.9	66.7	47.8
TD3+ACOACH	44.9 ± 28.5	27.8± 19.6	34.4	52.8	26.1 ± 35.3	20.3 ± 28.1	45.1	33.1
LCS	**21.7** ± **16.8**	**16.5** ± **9.2**	**19.2**	**27.3**	**17.7** ± **23.9**	**11.8** ± **17.4**	**29.6**	**21.27**

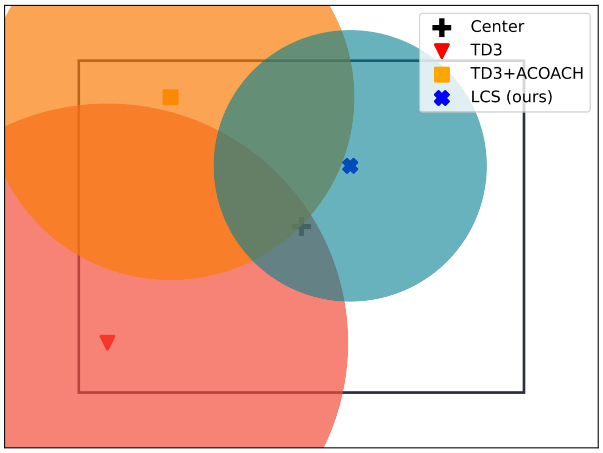


Fig. 13. Distribution of the landing points on the LP.

each algorithm's ability to track the UGV's movement over time.

As a comparison, the TD3 and TD3+ACOACH methods displayed poor tracking abilities in keeping up with the motion trend of the UGV, which could lead to speed up more to catch the UGV in landing (see Fig. 15(c) and 15(d)). This may also be one of the reasons for inaccurate landing under the control of these two algorithms.

D. Field Test

To evaluate the performance of our Landing Vision System (LVS) and Landing Control System (LCS) in real-world environments, we applied the optimal landing policies learned by the LCS to the real world. The quadrotor UAV S450, equipped with a Horizon Robotics Developer Kit X3[‡] (RDK X3) onboard computer and PX4 Autopilot flight controller was used for the tests. A USB camera was mounted perpendicular to the frame toward the ground. Similarly to the simulated training, the UAV's connection to the on-board camera remained constant.

In the field test, the UGV was permitted to move at random speeds, with a maximum of 0.5 m/s. We mounted a LP of 0.8 $m \times 0.6$ m on the back plate of the UGV. Although the AI acceleration method in RDK X3 was utilized, we still reduced the LVS's real-time detection frequency to reserve sufficient computational resources for the LCS-controlled UAV. The decision to lower the detection frequency of LVS was made to ensure adequate computational resources were available in real-time for the LCS controller. We conducted 5 landing trails in outdoor environments, and one of the landing results from a third person view is given in Fig. 17.

[‡]https://developer.horizon.ai

We highlighted some of the key waypoints of the UAV and UGV. The successful landing trajectory was similar to the trajectory in simulations. We observe that first, the LVS was able to identify the designated ground target using the YOLOv5 (see Fig. 18(b)).As the UGV was identified by the LCS and moved forward, the UAV could catch up with it at $t = 5s$, demonstrating its ability of tracking the UGV during landing. We then observed that at $t = 8s$, as the target UGV drive forward and the UAV lost track, the UAV quickly flew toward where the target had disappeared, as shown in Fig. 18(c). Furthermore, Fig. 18(d) demonstrates that at $t = 12s$, the UAV successfully regained tracking and continues the landing process. Furthermore, we reached an 80% success rate in the conducted experiments. The learned strategies were successfully implemented in real-world environments without requiring any parameter tuning, which verifies the stability of our method.

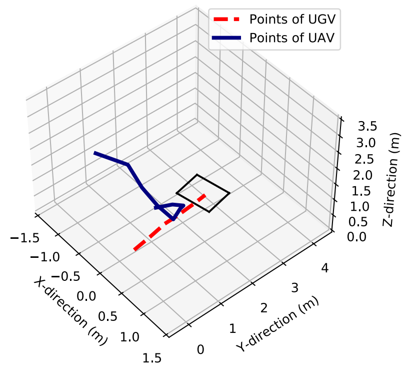
Furthermore, we reached an 80% success rate in the conducted experiments. The learned strategies were successfully implemented in real-world environments without requiring any parameter tuning, which verifies the stability of our method.

Before testing our landing approach, we evaluated the accuracy of our Depth Estimator (DE) network by comparing the predicted altitude with recorded barometer data. The results from both systems were comparable, as shown in Fig. 18. This demonstrated that the LVS was able to capture accurate information about the system state to be used to perform the autonomous landing task.

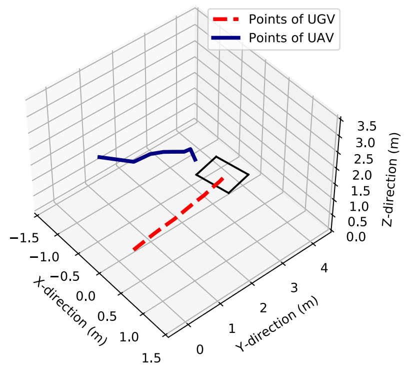
E. Discussion

Although the UAV can learn satisfying landing policies using the proposed method, there are still several issues worth discussing arising from the simulation and real-world experiments.

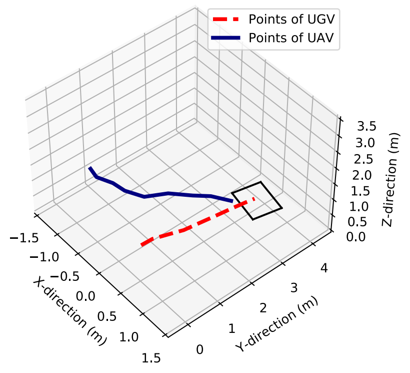
Tracking the unmanned ground vehicle (UGV) by the unmanned aerial vehicle (UAV) can be problematic even when the UGV moves in a constant low-speed, up to 1 m/s, and follows a fixed direction. In some instances, the UAV may lose track of the UGV even when it was about 3 m above the ground and it was assumed that the UGV was moving in a fixed direction. The detection was relatively stable when the UAV was at a distance from the UGV. However, when the UAV was getting closer, its field of view (FOV) would change quickly due to its sudden movement, making it difficult to locate the UGV again. Moreover, at such a close range, there was a risk that the UAV could not detect the marker at all, rendering it blind about its relative positioning to the UGV.



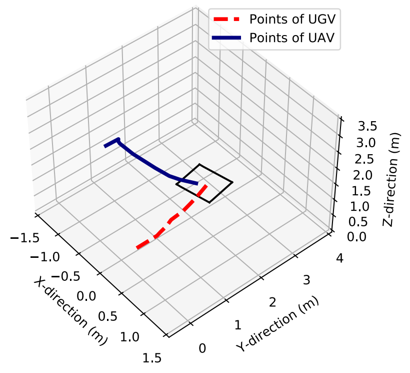
(a) A failure (TD3)



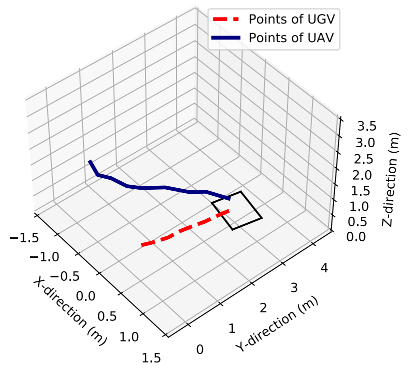
(b) A failure (TD3+ACOACH)



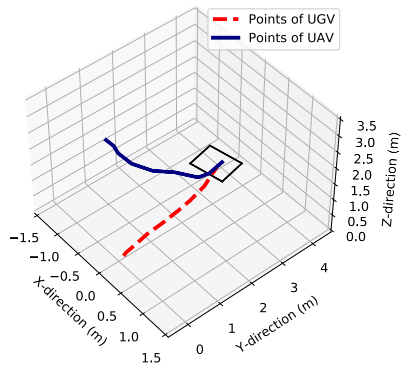
(c) A failure (LCS)



(d) A success (TD3)

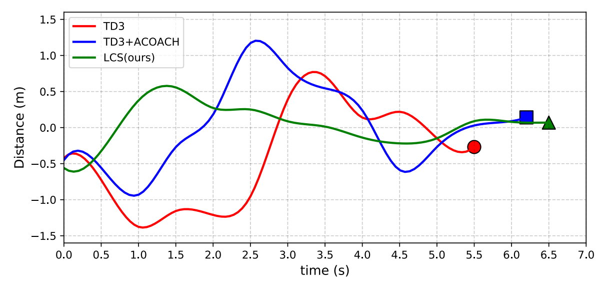


(e) A success (TD3+ACOACH)

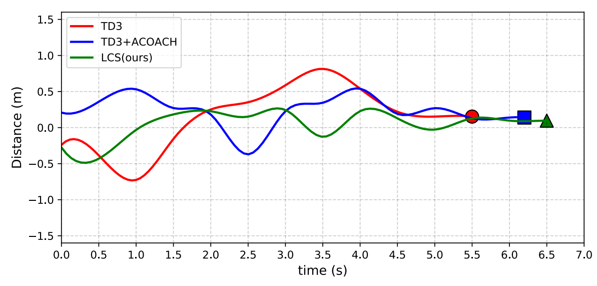


(f) A success (LCS)

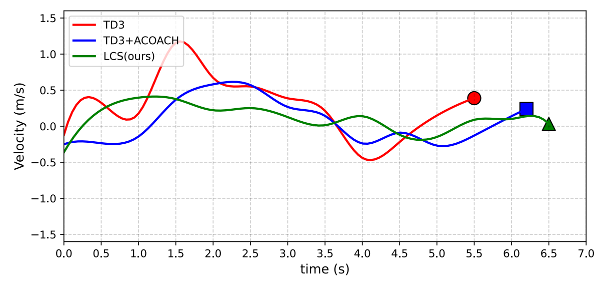
Fig. 14. Trajectories of successful and failed landing on a moving UGV ($v_{ugv} = 0.5\ m/s$) using three algorithms (TD3, TD3+ACOACH, LCS)



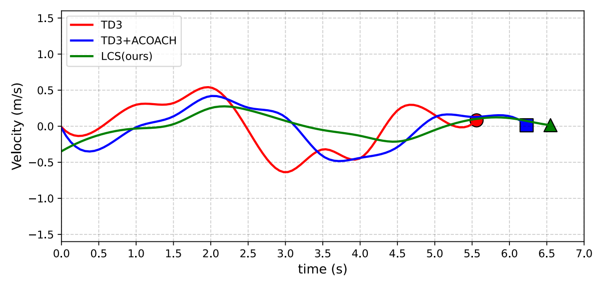
(a) Averaged position of the UAV with respect to the landing platform in the x-axes.



(b) Averaged position of the UAV with respect to the landing platform in the y-axes.



(c) Averaged velocity of the UAV with respect to the landing platform in the x-axes.



(d) Averaged velocity of the UAV with respect to the landing platform in the y-axes.

Fig. 15. Averaged partial state of 10 successful landing episodes for three algorithms in simulated testing phase under $v_{ugv} = 0.5\ m/s$.

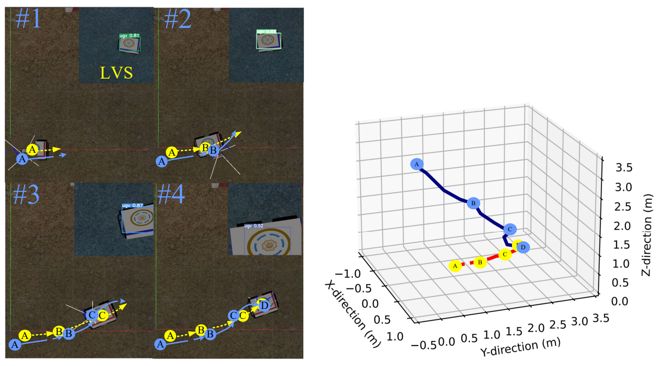


Fig. 16. Tracking and landing trajectories of the LCS-controlled UAV in the simulation, (v_{ugv} = 0.5 m/s)

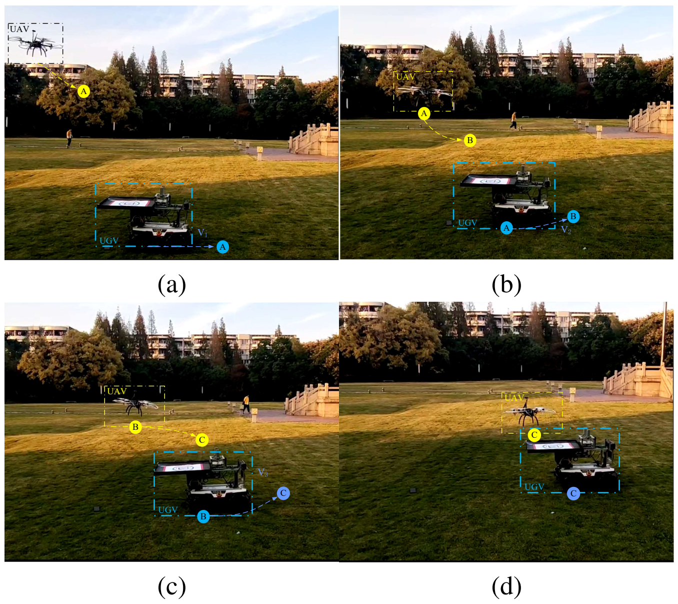


Fig. 17. Illustration of the landing trajectory using LVS and LCS in the field test with changing speed with a maximum velocity of 0.5 m/s.

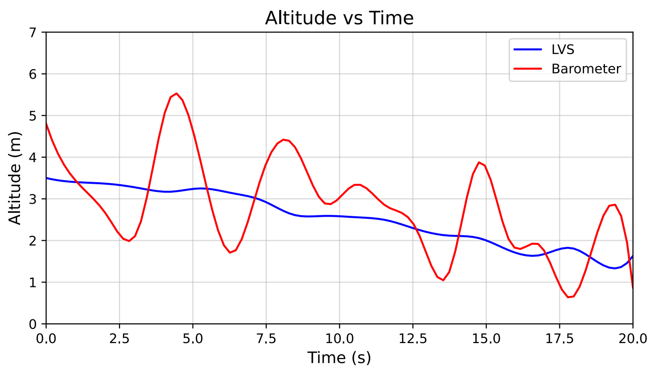


Fig. 18. The altitude information from the barometer was compared to a predetermined altitude for landing on the UGV, and to the height profile predicted by the DE algorithm.

In our training and testing, the UAV has learned to follow the UGV. Specifically, the UAV will accelerate towards the direction the UGV headed after leaving the UAV camera view. Two solutions could make tracking the UGV more achievable and consistent. First, one could use a wide-angle camera instead of an ordinary camera to enable the detection of the marker with higher precision. Second, the state of the UGV could be predicted, making use of the Extended Kalman filter [33]. By doing so, the UAV would track the UGV proactively, allowing it to move in a more consistent pattern, leading to more effective policy learning, and more competent policy use. Furthermore, this improved method should enable the UAV to learn more versatile landing skills to match the faster-moving UGV with more complicated movement trajectories.

In this paper, the state space was three-dimensional, considering only the 3D relative position between the UAV and the UGV. It would be more informative to include the relative heading direction of the UGV from the perspective of the UAV. If communication was allowed between the UAV and the UGV, not only would the heading direction be easily available, but also its speed for constructing the system state representation. Such information would be helpful for tracking the UGV in the FOV of the UAV. Besides, the altitude of the UAV could be considered for inclined quadrotor landing as discussed in [34].

The reward function was also essential for shaping the landing behavior of the UAV. In equation (7), a negative reward was given when the UAV approach the edge of the UGV in its FOV, which encouraged the UAV to keep tracking the UGV. If the tracking capability could be improved as discussed above, we would revise this reward because a good landing policy did not necessarily require sight of the UGV at every time step.

Due to the consideration of convenience and safety reasons, a training or testing episode in simulation was simply ended when the UAV reached 0.2 m vertically above the UGV. But the UAV was allowed to land on the UGV in the real-world experiment. Although the UAV demonstrated good performance in simulation, it happened occasionally that the UAV was likely to collide with the UGV at the end of the landing task in real-world. As the UAV did not learn how to land in the last 0.2 m, the UAV had to generalize the learned policy to account for this situation. However, it was essentially a blind landing strategy similar to using human intuition without seeing the state of the environment. This problem should be carefully considered from the perspective of sim-to-real transfer in reinforcement learning [35]. Specifically, the training experience can be enriched in simulation by taking into account the problems encountered in real-world. Continual and fast online learning methods can also be considered for adapting to unforeseen situations in real-world environments.

VII. Conclusion

In this paper, we have proposed a novel approach for autonomous UAV landing on a moving UGV. A vision system has been designed to detect the target marker and extract relevant information to estimate the relative position and velocity between the UAV and the UGV. The Landing Control System has been introduced to efficiently learn the optimal landing policy with human guidance. Our proposed algorithms has exhibited better performance in landing accuracy, time cost, and landing precision compared to benchmark deep reinforcement learning algorithms in simulation. Furthermore, the learned strategies were successfully implemented into real-world environments without requiring any parameter tuning. In future work, we aim to enhance the detection and tracking capabilities of the vision system to address the occasional problems of losing sight of the UGV. Additionally, we intend to design more curriculum-based experiments with increasing complexity to address the challenge of transferring the learned strategies from simulation to real-world environments.

References

[1] Pratap Tokekar, Joshua Vander Hook, David Mulla, and Volkan Isler. Sensor planning for a symbiotic uav and ugv system for precision agriculture. *IEEE Transactions on Robotics*, 32(6):1498–1511, 2016.

[2] Kevin Dorling, Jordan Heinrichs, Geoffrey G Messier, and Sebastian Magierowski. Vehicle routing problems for drone delivery. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(1):70–85, 2016.

[3] Hamid Menouar, Ismail Guvenc, Kemal Akkaya, A Selcuk Uluagac, Abdullah Kadri, and Adem Tuncer. Uav-enabled intelligent transportation systems for the smart city: Applications and challenges. *IEEE Communications Magazine*, 55(3):22–28, 2017.

[4] Michael A Goodrich, Bryan S Morse, Damon Gerhardt, Joseph L Cooper, Morgan Quigley, Julie A Adams, and Curtis Humphrey. Supporting wilderness search and rescue using a camera-equipped mini uav. *Journal of Field Robotics*, 25(1-2):89–110, 2008.

[5] Lizhen Wu, Chang Wang, Pengpeng Zhang, and Changyun Wei. Deep reinforcement learning with corrective feedback for autonomous uav landing on a mobile platform. *Drones*, 6(9):238, 2022.

[6] Long Xin, Zimu Tang, Weiqi Gai, and Haobo Liu. Vision-based autonomous landing for the uav: A review. *Aerospace*, 9(11):634, 2022.

[7] Muhammad Talha, Furqan Asghar, Ali Rohan, Mohammed Rabah, and Sung Ho Kim. Fuzzy logic-based robust and autonomous safe landing for uav quadcopter. *Arabian Journal for Science and Engineering*, 44(3):2627–2639, 2019.

[8] Yi Feng, Cong Zhang, Stanley Baek, Samir Rawashdeh, and Alireza Mohammadi. Autonomous landing of a uav on a moving platform using model predictive control. *Drones*, 2(4):34, 2018.

[9] Bora Erginer and Erdinc Altug. Modeling and pd control of a quadrotor vtol vehicle. In *2007 IEEE Intelligent Vehicles Symposium*, pages 894–899. IEEE, 2007.

[10] Khashayar Asadi, Akshay Kalkunte Suresh, Alper Ender, Siddhesh Gotad, Suraj Maniyar, Smit Anand, Mojtaba Noghabaei, Kevin Han, Edgar Lobaton, and Tianfu Wu. An integrated ugv-uav system for construction site data collection. *Automation in Construction*, 112:103068, 2020.

[11] Alejandro Rodriguez-Ramos, Carlos Sampedro, Hriday Bavle, Paloma De La Puente, and Pascual Campoy. A deep reinforcement learning strategy for uav autonomous landing on a moving platform. *Journal of Intelligent & Robotic Systems*, 93(1):351–366, 2019.

[12] Marwan Shaker, Mark NR Smith, Shigang Yue, and Tom Duckett. Vision-based landing of a simulated unmanned aerial vehicle with fast reinforcement learning. In *2010 International Conference on Emerging Security Technologies*, pages 183–188. IEEE, 2010.

[13] Seongheon Lee, Taemin Shim, Sungjoong Kim, Junwoo Park, Kyungwoo Hong, and Hyochoong Bang. Vision-based autonomous landing of a multi-copter unmanned aerial vehicle using reinforcement learning. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 108–114. IEEE, 2018.

[14] Jingyi Xie, Xiaodong Peng, Haijiao Wang, Wenlong Niu, and Xiao Zheng. Uav autonomous tracking and landing based on deep reinforcement learning strategy. *Sensors*, 20(19):5630, 2020.

[15] Chang Wang, Jiaqing Wang, Changyun Wei, Yi Zhu, Dong Yin, and Jie Li. Vision-based deep reinforcement learning of uav-ugv collaborative landing policy using automatic curriculum. *Drones*, 7(11):676, 2023.

[16] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *In Proceedings of 4rd International Conference on Learning Representations (ICLR)*, 2016.

[17] Sara Minaeian, Jian Liu, and Young-Jun Son. Vision-based target detection and localization via a team of cooperative uav and ugvs. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 46(7):1005–1016, 2015.

[18] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[19] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[20] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[21] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[22] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[23] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[24] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[25] Riccardo Polvara, Sanjay Sharma, Jian Wan, Andrew Manning, and Robert Sutton. Autonomous vehicular landings on the deck of an unmanned surface vehicle using deep reinforcement learning. *Robotica*, 37(11):1867–1882, 2019.

[26] Hado Van Hasselt and Marco A Wiering. Using continuous action spaces to solve discrete problems. In *2009 International Joint Conference on Neural Networks*, pages 1149–1156. IEEE, 2009.

[27] Carlos Celemin and Javier Ruiz-del Solar. An interactive framework for learning continuous actions policies based on corrective feedback. *Journal of Intelligent & Robotic Systems*, 95:77–97, 2019.

[28] Jorge Ramírez, Wen Yu, and Adolfo Perrusquía. Model-free reinforcement learning from expert demonstrations: a survey. *Artificial Intelligence Review*, 55(4):3213–3241, 2022.

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *In Proceedings of 3rd International Conference on Learning Representations (ICLR)*, 2015.

[30] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, Georgia, USA, 2013.

[31] Thien-Minh Nguyen, Thien Hoang Nguyen, Muqing Cao, Zhirong Qiu, and Lihua Xie. Integrated uwb-vision approach for autonomous docking of uavs in gps-denied environments. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9603–9609. IEEE, 2019.

[32] Jiang Zhao, Han Liu, Jiaming Sun, Kun Wu, Zhihao Cai, Yan Ma, and Yingxun Wang. Deep reinforcement learning-based end-to-end control for uav dynamic target tracking. *Biomimetics*, 7(4):197, 2022.

[33] Maria Isabel Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties. *Institute for Systems and Robotics*, 43(46):3736–3741, 2004.

[34] Jacob E Kooi and Robert Babuška. Inclined quadrotor landing using deep reinforcement learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2361–2368. IEEE, 2021.

[35] Hao Ju, Rongshun Juan, Randy Gomez, Keisuke Nakamura, and Guangliang Li. Transferring policy of deep reinforcement learning from simulation to reality for robotics. *Nature Machine Intelligence*, 4(12):1077–1087, 2022.