IEEE Robotics and Automation Letters (RAL) paper presented at
2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Autonomous Landing on a Moving Platform Using Vision-Based Deep Reinforcement Learning

Pawel Ladosz, Meraj Mammadov, Heejung Shin, Woojae Shin, and Hyondong Oh, *Senior Member, IEEE*

***Abstract*—This letter describes autonomous landing of an unmanned aircraft system on a moving platform using vision and deep reinforcement learning. Landing on the moving platform offers several benefits, such as more mission flexibility and reduced flight time. In particular, the end-to-end vision approach (i.e., an input to the reinforcement learning is a raw image from the camera) with the deep regularized Q algorithm and custom designed reward is utilized. The custom reward was specifically devised to encourage useful feature extraction from the state space. Additionally, the proposed reinforcement learning algorithm has full 3D velocity control including the vertical channel. The simulation results show that the proposed approach can outperform existing approaches which use high-level extracted features (such as relative position and velocity of the landing pad). The simulation results are then successfully transferred to the real-world experiment by utilizing domain randomization.**

***Index Terms*—AI-enabled robotics, aerial systems: Applications, reinforcement learning, vision-based navigation.**

I. Introduction

UNMANNED aircraft systems (UASs) have the potential to assist with tedious tasks such as deliveries, environment monitoring and safety inspections. At the end of any mission, UASs need to land safely. In some cases, to save time and energy, it is beneficial for the UAS to be able to land on a moving platform. For example, during package delivery, the landing platform could move across the delivery zone while the UAS need to freely take off and land to deliver packages. In this paper, we address the issue of landing on a moving platform using vision-based deep reinforcement learning. Vision-based landing approaches have several advantages: (i) sensors (cameras) are inexpensive; (ii) cameras are widely available on UASs; and (iii) they require minimal investment for the landing pad and infrastructure.

There exist several attempts in the past which aim to address the landing on the moving platform using visual servoing and reinforcement learning methods. Visual servoing approaches can be generally split into position-based visual servoing (PBVS) [1] and image-based visual servoing (IBVS) [1]. The PBVS-based algorithm utilizes the 3-D position error based on the geometrical properties of the target landing pad, while the IBVS uses the 2-D pixel error. For the PBVS method, one of the key considerations is the accuracy of position estimation and control signal. One approach which shows good estimation accuracy of the position was devised by Yang et al. [2] where the vision estimate was fused with inertial measurement unit (IMU) data using unscented Kalman filter. The use of more accurate pan-tilt based visual servoing (PTBVS) [3] was proposed in [4]. The accuracy of the control signal was improved in Zhao et al. [5], where time-varying uncertainty (e.g., due to increased controller computation) to produce more time-appropriate commands was considered. Despite those advances, PBVS-based approaches can still suffer from large estimation errors, due to camera measurement errors [6].

For this reason, IBVS methods were devised, which rely on image features. Some early IBVS work includes [7] and [8]. Herisee et al. [7] proposed one of the first IBVS approaches for autonomous landing while Lee et al. [8] extended it to account for the ground effect. To reduce computational costs, Borshchova et al. [9] proposed colour-based IBVS, instead of traditional feature extractors. Cho et al. [10] used the Kalman filter to better estimate the forward velocity of the moving and oscilating landing platform, which was used as the feed-forward control term along wiht the IBVS control. Both IBVS and PBVS have several limitations, namely the need of utilizing special landing pads, extracting hand-designed features and relying on expensive and time-consuming tuning. To mitigate these issues, reinforcement learning-based approaches were devised.

In general, reinforcement learning approaches used for landing on the moving platform are limited to those utilizing the state information extracted from images such as positions and velocities in relation to the landing pad. One of the most recent approaches in this area was devised in [11], [12], [13]. In this work, Deep Deterministic Policy Gradient (DDPG) is used to land the UAS on a moving platform with the extracted relative

Manuscript received 19 October 2023; accepted 4 March 2024. Date of publication 20 March 2024; date of current version 5 April 2024. This letter was recommended for publication by Associate Editor P. Falco and Editor J. Kober upon evaluation of the reviewers' comments. This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant 2020R1A6A1A03040570, in part by the National Research Foundation of Korea (NRF) funded by the Korea Government under Grant 2023R1A2C2003130, and in part by Unmanned Vehicles Core Technology Research and Development Program through the National Research Foundation of Korea (NRF), Unmanned Vehicle Advanced Research Center (UVARC) funded by the Ministry of Science and ICT, the Republic of Korea under Grant 2020M3C1C1A01082375. *(Corresponding author: Hyondong Oh.)*

Pawel Ladosz is with the Department of Mechanical, Aerospace and Civil Engineering, University of Manchester, M13 9PL Manchester, U.K. (e-mail: pawel.ladosz@manchester.ac.uk).

Meraj Mammadov, Heejung Shin, Woojae Shin, and Hyondong Oh are with the Department of Mechanical Engineering, Ulsan National Institute of Science and Technology, Ulsan 44610, South Korea (e-mail: meraccos@gmail.com; godhj@unist.ac.kr; oj7987@unist.ac.kr; h.oh@unist.ac.kr).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2024.3379837, provided by the authors.

Digital Object Identifier 10.1109/LRA.2024.3379837

position and velocity. Another example of an approach utilizing positions and velocities was shown in [14]. It is worthwhile noting that, end-to-end (i.e., without using the extracted state information explicitly) vision-based reinforcement learning methods are currently limited to stationary landing platforms only. One example of such an approach was shown in [15] and [16]. This approach consists of two separately trained reinforcement learning agents: high-level to bring the UAS to the area above the landing pad and low-level to perform the landing. Additionally, domain randomization was used to bridge the sim-to-real gap. This approach was later extended in [17] and [18] with better generalization capability. Xu et al. [19] used double Q-learning to land on a stationary platform using vision only. The problem of landing on the inclined platform was considered in [20].

Thus far, deep reinforcement learning approaches are mostly used with relative state information (such as position or velocity). The ones that use raw images are restricted to stationary landing platforms. Moreover, vertical velocity control is usually given to the separate rule-based algorithm. Here, we propose an end-to-end vision-based deep reinforcement learning approach for landing on a moving platform without extracted state information. The input to the reinforcement learning is the output from the camera and UAS's inertial measurement unit (IMU) such as velocity and attitude. End-to-end approach offers two main advantages. First, it utilizes a singular algorithm, which saves integration efforts. Second it can be more robust to failure, as it can learn how to cope with situation where failure of one part of the algorithm would cause catastrophic failure of overall algorithm. To allow end-to-end learning with deep reinforcement learning, two techniques are employed in this study. First, we utilize the state-of-the-art deep regularized Q version 2 (DrQv2) reinforcement learning algorithm [21]. DrQv2 was designed explicitly to be as sample efficient as possible in robotics problems. Second, we design a new reward function, which minimises negative stimuli (reward) to the agent, while rewarding the agent for keeping the landing pad within the line-of-sight (LOS) of the UAS. This should help with exploration as has been shown in [22]. Also, the reward is constructed to allow landing with vertical velocity controlled by the reinforcement learning algorithm. The performance of the proposed method is verified in extensive simulations as well as real-world experiments using domain randomization.

Main contributions of our work can be summarized as follows:

- We present the first end-to-end vision-based deep reinforcement learning framework that directly utilizes pixels as an input to the network instead of a hand-crafted feature extractor;
- The proposed approach is compatible with existing landing infrastructure since it can be trained to handle arbitrary landing pad designs;
- We allow the reinforcement learning agent to control both horizontal and vertical velocity to give it ability to slow down and position itself better for landing; and
- We utilize domain randomization methods to allow landing on arbitrary and unseen backgrounds.

The rest of the paper is organized as follows. In Section II the details of the problem are described. Section III describes the DrQv2 algorithm, state and action space. In Section IV, the details of the reward implementation are outlined. Section V shows the simulation and experimental results. Finally, in Section VI conclusions and future work are presented.

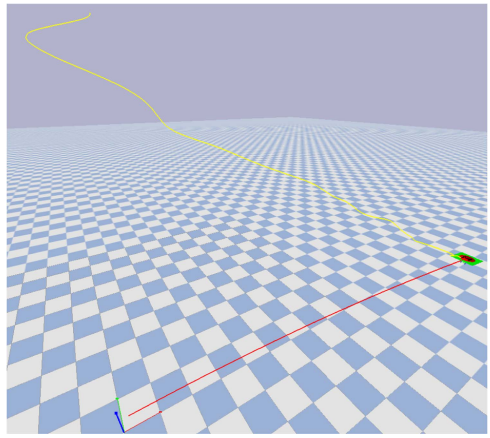

Fig. 1. Example of a successful landing trajectory. The UAS path is represented in yellow while that of the landing platform is in red.

II. Problem Formulation

This work considers autonomous landing of a UAS on a moving platform using vision information only. The landing platform is moving forward with velocities of up to 1 m/s. The landing platform has a dimension of 1 by 1 meters, with a circle with the letter 'H' in the middle (see Figs. 1 and 8 for the scenario depiction). Note, the particular design of the landing platform is not important, as long as the agent has seen it in the training and it has some distinguishable features. The letter H was chosen as it is used for helipads in general. The landing is considered successful when: (i) the full UAS is within the platform boundary; and (ii) the UAS touches down on the platform.

This work utilizes CrazyFlie UAS from the gym pybullet drones library [23]. CrazyFlie is a small vehicle, and thus, it is easy and safe to use allowing quick experimental iterations. The UAS is assumed to have a non-gimballed camera looking downwards. Moreover, the UAS is assumed to be able to position itself 10 meters over the landing platform with +/−3 m accuracy (which represents typical GPS accuracy). The UAS speed is limited to 1.5 m/s horizontally and 0.5 m/s vertically. It is assumed that the landing platform and the UAS cannot communicate or exchange any information. This mitigates the need for reliable communication and reduces landing platform costs. Note those restrictions are mostly due to limitations of the platform for the experiment, rather than limitations of the algorithm itself. Fig. 2 shows the model of the ground vehicle and UAS.

The landing problem is formulated as a continuous Markov decision process (MDP) with the following tuple (S, A, P, R, γ) where S is a set of continuous states and A is a set of continuous actions (UAS control inputs). P is a transition probability between states as:

$$p\left(s_{t+1}|s_1, a_1, s_2, a_2, \ldots, s_t, a_t\right) = p(s_{t+1}|s_t, a_t), \quad (1)$$

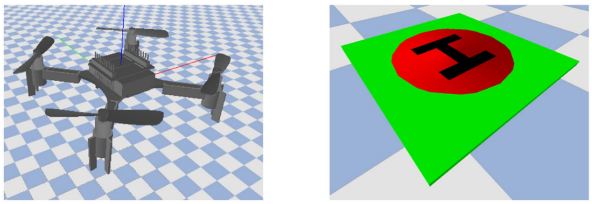

Fig. 2. Model of the UAS and the landing pad on a ground vehicle used in this study.

R is a set of rewards, and $\gamma \in (0, 1]$ is a discount factor. The task of the agent is to maximize the reward by selecting the best possible actions $a_t \in A$ in a step t, given the state $s_t \in S$ from the environment. Formally, this is defined as:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{x\sim P}\left[\sum_{t=0}^{\infty}\left(\gamma R(s_t, a_t, s_{t+1})\right)\right] \tag{2}$$

where π^* is the optimal policy and t is the timestep.

III. Reinforcement Learning Agent

Reinforcement learning is one of the families of techniques used for solving MDP problems. For an overview and explanation of reinforcement learning, the reader is invited to [24], [25]. In reinforcement learning methods, the algorithm is trained by continuously interacting with the environment and assessing its performance in terms of the reward.

In general there exist several reinforcement learning methodologies such as policy-based and value-based approach. Here, a method called actor-critic is utilised, where two networks are trained. The first one, called the actor, makes decisions, and the second one, called the critic, judges the quality of that decision. Using the actor-critic framework has been shown to provide outstanding performance and stability during learning [24]. There exist several variants of actor-critic approaches, and here, an approach called Deep Regularized Q version 2 (DrQv2) [21] is utilised.

A. Deep Regularized Q Version2

Continuous control from visual information is generally challenging [21]. One of the challenges is learning state representations in a continuous domain due to the number of possible states. Deep Regularized Q version2 (DrQv2) was explicitly designed to help address this issue. While it is using deep deterministic polict gradient (DDPG) [26] as a baseline learning method, it adds image augmentation to improve the performance. In particular when a stack of images is sampled from experience replay, they are shifted in random directions by 4 pixels, while remaining pixels are filled by zero padding. This augmentation yields significant performance gains both in terms of sample efficiency and performance [21]. Additionally several other techniques, such as the usage of bilinear interpolation and extremely fast replay buffer implementation, makes this approach particularly appealing in terms of performance and speed.

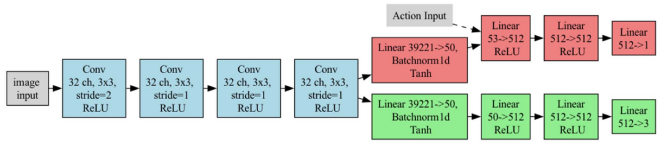

Fig. 3. Architecture of the network. The red, green, and blue color represent critic, actor, and image encoder, respectively.

B. Deep Deterministic Policy Gradient

To address the challenges of continuous control in reinforcement learning, DDPG was proposed by [26]. One of those challenges is the strong correlation between subsequent states, which can introduce substantial bias during training. To address this issue, DDPG uses experience replay, proven effective in the DQN network [27]. In experience replay, a finite-length queue stores past experiences, and the network is trained on random batches drawn from this queue.

Another training challenge for RL networks is the potential instability and divergence that occur when the same network is used both to optimize the policy and for bootstrapping. DDPG addresses this issue by using a second, target network for the critic, which is softly updated towards the training network at each optimization iteration.

The critic estimates the value of the states, by using a variant of the Bellman equation, with its loss function can be formulated as:

$$L(\theta^Q) = \mathbb{E}_{\mu'}\left[\left(Q\left(s_i, a_i|\theta^Q\right) - y_i\right)^2\right], \tag{3}$$

where

$$y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1}|\theta^{\mu'}\right)|\theta^{Q'}\right). \tag{4}$$

Here, Q' represents the output value from the target critic network, and μ' is the output action from the target actor network.

The actor network is optimized according to policy-based methods, with its gradient expressed as:

$$\nabla_{\theta^\mu} J \approx \mathop{\mathbb{E}}_{s_t\sim\rho^\beta}\left[\nabla_a Q(s, a|\theta^Q)\Big|_{\substack{s=s_t,\\ a=\mu(s_t)}} \nabla_{\theta_\mu}\mu\left(s|\theta^\mu\right)\Big|_{s=s_t}\right], \tag{5}$$

where $\nabla_a Q(s, a|\theta^Q)$ is the gradient of the Q-value with respect to the action. The chain rule is applied to calculate the gradient of the objective function with respect to the actor parameters, hence the product of the two gradients. The expectations are approximated using mini-batch updates from experience replay. The action a is chosen by the actor network, $\mu(s_t|\theta^\mu)$, for the current state s_t.

The network architecture is outlined in Fig. 3.

C. State and Action Space

The state consists of a greyscaled image from a downward-facing camera as shown in Fig. 4 and velocity and attitude of

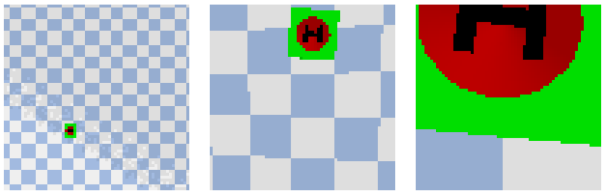

Fig. 4. Image input to the neural network (before greyscaling) at various stages of landing.

the UAS. The attitude and velocities are readily available from UAS's autopilot. The action output of the actor is used to control the velocity. For the actor convergence, actions and states are normalized between 0 and 1. Then, the velocity command is obtained:

$$\mathbf{v}_v = 2.0 \cdot (\mathbf{v}_c - 0.5) \cdot \mathbf{v}_l \tag{6}$$

where $\mathbf{v}_v$ (m/s) is the velocity vector, $\mathbf{v}_c \in \{0, 1\}^3$ is the neural network output, and $\mathbf{v}_l$ is a vector representing the UAS's velocity limits in each dimension. In our case, we use 1.5 m/s for the horizontal velocity limit and 0.5 m/s for the vertical velocity limit.

D. Domain Randomization

Due to the computational speed PyBullet [28] was selected as the simulation environment. One of the popular available UASs in this simulator is Crazyflie 2.1. Naturally, the simulation is imperfect reflections of real life with several inaccuracies such as physics and lighting. Normally those inaccuracies are grouped under sim-2-real gap term. To close the sim-2-real gap several techniques exists such as domain randomization, generative networks or improving the accuracy of the simulator. In this work domain randomization technique was utilized, in the form of several separate randomizations outlined below.

- To make the agent robust to different backgrounds and irrelevant visual disturbances, about 400 textures were chosen and randomly applied on the floor at every episode. The textures came from dtd data set (https://www.robots.ox.ac.UK/vgg/data/dtd/).
- To account for changes in the ambient lighting conditions, a randomly chosen scaling factor $\in \{0.5, 1.5)\}$ was applied on the image input to the network in the simulation. This also helps account for various shadow and lighting conditions present during the landing.
- To allow UAS's to land from different relative positions to the landing pad and to simulate the GPS error, the UAS is initialized with up to 3 m offset in each direction at 10 m above the landing pad.
- To allow landing with various landing platform velocity, the speed of the ground vehicle was randomly picked from a range of {0.0, 1.0} m/s at every episode. To account for imperfect alignment of the UAS and the ground platform, the initial heading of the vehicle is randomly picked from a range of $\{-15, 15\}$ degrees.
- To account for the occurance of physical distractions in real experiments, random 3D objects of different shape and sizes were added to the simulation ground.
- Unlike the previous work [17], [18], the real image of the landing pad was supplied to the simulator. This significantly accelerated convergence and increased robustness with landing pad detection during experiments. Simulating realistically looking landing pads in PyBullet is very challenging due to the limited lighting and textures simulation. The images were taken from 2 meters height and the landing pad was cropped out and imprinted on the simulated pad. While this may seem like oversimplification, we argue that, in the case of the real landing, the shape and design of the landing pad is known beforehand and pictures of it can be taken, and thus in practice this assumption is justified.
- To make the agent robust to various delays in the controller and image transfer, the simulation's step length is randomized by sampling it from a range of {0.1, 0.4} seconds. In other words, the neural network received an image and made a decision every 0.1 to 0.4 seconds.

IV. Reward

For reinforcement learning-based landing, it is necessary to design a suitable reward function. The design was guided by the following principles:

- Land as fast and safely as possible to reduce the overall mission time and energy usage;
- Land as close to the centre of the landing pad as possible to reduce the landing platform size and for safety;
- Minimise reward parametrization for easier tuning.; and
- Avoid exceeding the safe descent velocity for safety.

With those in mind, the final reward takes the form of:

$$R_t = \begin{cases} \beta R_z + \alpha R_{xy} + \gamma R_l, & \text{if landing platform visible} \\ R_p, & \text{otherwise} \end{cases} \tag{7}$$

where R_t is the total reward given to the agent per step, R_{xy} is the reward for reducing the horizontal distance error, R_z is the reward for appropriate z-velocity, R_l is a termination reward for landing or crashing and R_p is a punishment for non-beneficial behaviours. Each element of the reward will be described in greater details below.

A. Landing Platform Visibility

The landing platform visibility in the reward is realized by the if-statement in (7). The agent is only able to obtain all the other rewards if the landing platform is within the LOS; otherwise, it is punished. In essence, this is a binary 'switch' to all other rewards. This reward contributes to extracting features from the landing pad faster, as the agent should quickly learn the importance of the landing platform. It also enables recovery behaviours when the landing pad is not within the LOS of the UAS, since the agent can only receive a positive reward if it finds the landing platform again.

B. Vertical Velocity

For the agent to land, it is necessary to include a reward for descending. Normally, the vertical distance is used, however, it suffers from several issues. First, when the positive reward is given based on the vertical distance, the UAS tends to hover just above the landing pad without landing. This behaviour happens as the UAS is trying to collect all the rewards associated with being close to the landing pad. On the other hand, if the agent is punished for the excessive height, it can lead to UAS crashing into the landing pad to stop the punishment. To mitigate those issues either extensive tuning is required or, more commonly, the vertical velocity is controlled by the external algorithms [11], [12], [13].

Here, a reward is proposed, which allows vertical control directly. Specifically, the reward is given for descending at or below the safe velocity. The reward is constructed as an exponentially growing reward up to the maximum safe descent velocity. This reward prevents two aforementioned issues as: (i) the UAS will not be able to hover over the landing platform; and (ii) the UAS should never exceed the safe velocity, and thus landing pad crashing is also prevented. This component of the reward is formalized as:

$$R_z = \begin{cases} R_p & \text{if } \mathrm{v_c} < \mathrm{v_l or v_c} > 0 \\ \frac{\zeta^{(v/v_l)}-1}{\zeta-1} & \text{otherwise} \end{cases} \tag{8}$$

where ζ is a parameter determining how steeply a reward increases as the UAS gets closer to the limit of safe velocity, V_l is the negative safe descent velocity limit, v is the UAS's current vertical velocity and v_c is the commanded vertical velocity by the network.

C. Horizontal Distance

To control the UAS horizontally the reward is given for keeping the UAS as close to the middle of the landing platform as possible. This should also encourage the UAS to match the velocity of the pad; otherwise, the maximum reward cannot be achieved. Similar to the vertical velocity reward component, the horizontal distance reward is scaled exponentially with the horizontal distance to the pad. This element of the reward can be formally defined as:

$$R_{xy} = \begin{cases} 0 & \text{if } \mathrm{d_{xy}} > \mathrm{d_s} \\ \frac{\rho^{(d_s - d_{xy}/d_s)}-1}{\rho-1} & \text{otherwise} \end{cases} \tag{9}$$

where ρ is an parameter determining how steeply reward increases as the UAS gets closer to the landing platform, d_s is a parameter to determine at which distance to start giving a positive reward to the agent and d_{xy} is current UAS's horizontal distance to the landing platform centre.

D. Landing/crashing

The final component determines the amount of reward when the UAS lands on the platform or crashes to the ground. The reward for successful landing is highly positive while for crashing it is slightly negative. It may be tempting to provide a large negative reward upon crashing, however, this would likely lead to undesired side effects. It is important to note that, at the beginning of the training, successful landings are very rare. Thus, the UAS is unlikely to learn the benefits of the landing and instead it would learn that descending results in the negative reward. Then, if the crashing reward is highly negative, the UAS would be reluctant to descent, preventing learning to land. This can be summarized as follows:

$$R_l = \begin{cases} R_s & \text{if landed} \\ R_p & \text{if crashed} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

where R_s is a parameter which determines how much the agent is rewarded for successfully landing.

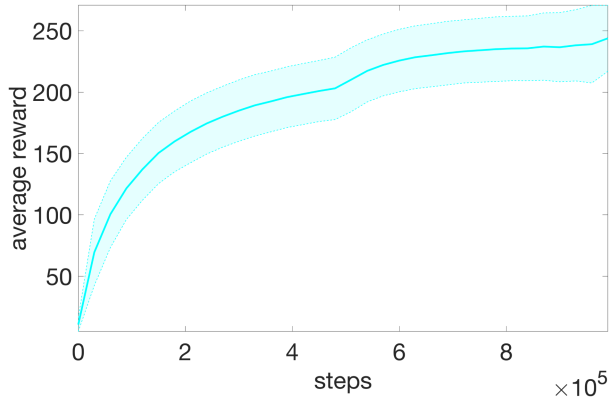

Fig. 5. Learning curve of the proposed approach.

V. Results

The result section aims to answer the following questions:

- Does the proposed algorithm learn and perform as quickly and safely as possible?
- Does the proposed reward contributes to learning with visual information?
- Is the performance comparable with other approaches?
- Is performance loss due to the domain randomization sufficiently small?
- Is domain randomization sufficient for the proposed approach to work on arbitrary surfaces?

A. Simulation Results

All simulations are performed and averaged across 5 random seeds. After training was completed, for each approach and seed, the most scoring, network was selected for evaluation of all simulation and experimental scenarios. The UAS is initialized at 10 meters above the landing platform with the uniform noise of ± 3 meters. The landing platform moves forward with the velocity between 0.0 m/s and 1 m/s. The episode is terminated under three conditions: (i) the UAS has landed, (ii) the UAS has crashed to the ground or (iii) the maximum number of steps has been reached. The training, simulation and scenario parameters are listed in Table I.

1) Performance of the Proposed Algorithm: To show the learning behaviour, the learning curve is presented in Fig. 5.

TABLE I
SIMULATION PARAMETERS

Parameter	Value
Scenario parameters	
Episode length	30 seconds (900 steps)
Episode number	4000
Random seeds	$1, 2, 3, 4, 5$
Aggregated steps	10
Safe descent velocity	$-0.5m/s$
DrQv2 parameters	
Frame stack	3
Image resolution	84 by 84 pixels
Learning rate	0.005
Soft update parameter (τ)	0.0012
Automatic entropy tuning	yes
Batch size	1024
Replay size	300000
Discount factor (γ)	0.99
Warm-up steps	10000
Reward parameters	
Punishing reward R_p	-0.01
Rewards weighing factors (α, β, γ)	$(0.4, 1, 1)$
Vertical velocity parameter ζ	30
Horizontal distance parameter ρ	20
Horizontal threshold of positive reward d_s	$10m$
Landing reward	120

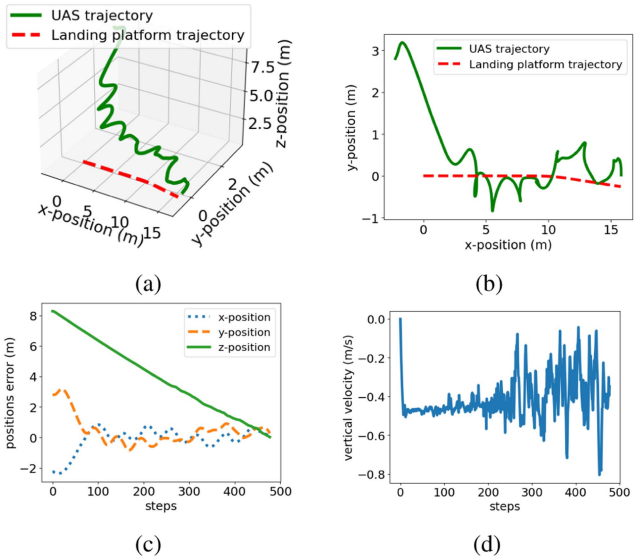

Fig. 6. Example trajectory of the proposed approach: (a) shows the trajectory taken by the UAS and the landing platform; (b) shows the top view of the trajectory taken by the UAS and the landing platform; (c) shows the position error between the UAS and platform; and (d) shows the vertical velocity of the UAS.

The results are averaged over 5 random seeds and the shaded area represents a boundary with one standard deviation. During training, evaluation is performed over 10 episodes every 100 episodes.

The sample scenario and trajectories are outlined in Fig. 6. The proposed approach learned to fly at safe velocity. Note that the sudden increase in the velocity just before the landing pad is to help the UAS to cope with the ground effect. The trajectory exhibits a constant descent while trying to keep the landing pad underneath the UAS.

TABLE II
PERFORMANCE COMPARISON OF VISION DRQV2 WITH OTHER STATE-OF-THE-ART AUTONOMOUS LANDING ALGORITHMS

Approach	Variant	Success rate	Landing error
Vision DrQv2	No randomization (Ours)	97.4%	0.52
	Randomization (Ours)	91.8%	0.63
	No LOS	82.4%	1.78
	Constant descent velocity	75.2%	2.02
FF-IBVS	External noise	65.7%	0.61
	No noise	87.8%	0.58
DDPGLN	External noise	31.6%	1.64
	No noise	75.4%	0.64
Vision SAC	N/A	60.1%	0.75

2) Comparison With Other Algorithms: The proposed algorithm is compared against: (i) feed-forward image-based visual servoing (FF-IBVS) [10], (ii) deep reinforcement learning approach from [13] (DDPGLN) and (iii) exactly the same as the proposed approach but using SAC instead of DrQv2 (called the Vision SAC hereafter). For both non-vision baselines, two methods were tested with no noise and with an external noise. For external noise, landing pad state estimation (i.e., velocity and positions) was fed with the extra randomly-generated errors of up to $\pm 10\%$. This was done to emulate errors resulting from estimators such as the Kalman filter [10]. For DDPGLN, hyperparameters were reused from [13], with some retuning to improve the performance on our scenario. Note that, since the proposed approach implicitly extracts the information about the landing pad state from vision, it is not affected by this noise. Additionally, the proposed approach is compared against itself but without the landing platform visibility component of the reward (called vision DrQv2 with no LOS hereafter). This comparison serves to show the importance of the landing platform visibility reward to the overall performance. Note, to save computational efforts, simulations in this section are performed with a single background and single image of the landing pad without any other domain randomization.

The comparison is performed using 500 evaluation episodes and the averaged result is presented in Table II. The proposed approach shows the best performance. DDPGLN shows a lower performance than that reported in [13], which is due to two reasons: (i) the scenario tested here is more complex with the UAS starting further away and higher than the ones tested in [13]; and (ii) the original experiments utilizes overly accurate state estimation such as Mocap positioning system or AruCo marker-based system which can provide sub centimeter accuracy if well tuned. Once we add enough external noise to the state estimator to make it more akin to the real world, it fails to hold up its original performance (see external noise results in Table II). FF-IBVS is struggling to keep up with the speed of the ground vehicle without overshooting, and thus its success rate is also low.

Comparing the proposed approach with vision DrQv2 with no LOS, it can be noted that without the landing platform visibility reward, the performance degrades. The most likely reason for this behaviour is that the UAS is collecting the reward for descending and rarely discovers a high reward for landing on the platform. While not conclusive, this supports the idea that

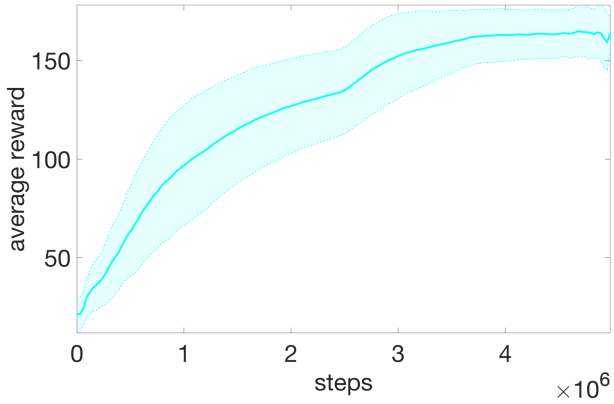

Fig. 7. Learning curve of our approach with domain randomization.

the landing platform visibility component helps reinforcement learning to understand the importance of the landing pad quicker. Also, comparing the performance between using a constant vertical velocity and allowing an agent to control the velocity, it can be seen that the latter performs better. This is mostly that at high landing platform speed, the agent with the constant vertical velocity is not able to slow its decent rate to the keep landing pad in-sight before being able to fly over it in the x-y plane.

3) Performance With Domain Randomization: Domain randomization is necessary to close the sim-2-real gap, however too much randomization can have significant impact on performance. Thus it is necessary to utilize enough domain randomization to allow experiment but not too much as to not to loose too much performance. This is summarized in Table II. It can be seen that with domain randomization performance is reduced by 6%. This is acceptable performance loss for what becomes a much more complex problem. The complexity arises due to i) increased difficulty of distinguishing landing pad from the background; ii) more directions the landing pad can move relative to the UAS; and iii) random response time for control commands. The converged learning curve is shown in Fig. 7. Note that, with domain randomization, convergence is much slower (1 million steps compared with 5 million steps) due to complexity.

B. Experimental Results

In this section experiment results on the real vehicle are described.

1) Experimental Setup: In this experiment CrazyFlie 2.1 vehicle was utilized. This vehicle offers several advantages. First, as it is small, tests are easy to perform. Second, it is robust to crashes. The vehicle has the camera-equipped AI-deck and lighthouse deck to give position and velocity estimation capability. Note that we only use the position for data collection purposes. The vehicles used in this experiment is depicted in Fig. 8.

The ground vehicle used is a Turtlebot 2 with a landing pad attached to it. The picture of the ground vehicle with the landing pad is shown in Fig. 8. The experiment is performed in a room which was previously unseen during training, in terms

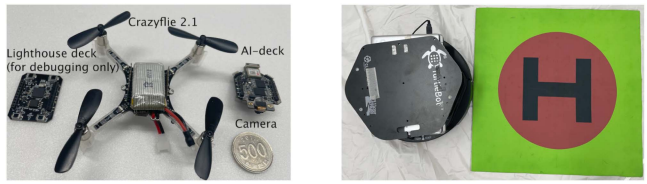

Fig. 8. UAS and ground vehicle used for experiment.

Fig. 9. Experimental area with two different backgrounds.

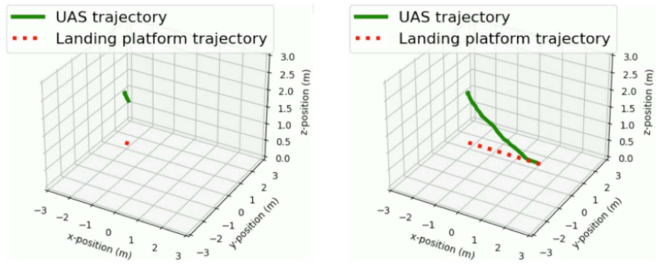

Fig. 10. Example experimental run, episode start ($t = 1\ sec$) and end ($t = 6\ sec$).

TABLE III
SUMMARY OF EXPERIMENTAL RESULTS WITH DIFFERENT LANDING PAD VELOCITIES

Speed (m/s)	Success rate (10 trials)	
	Background 1	Background 2
0.0	100%	100%
0.25	80%	80%
0.5	80%	70%
0.75	70%	80%
1.0	70%	70%

of lighting, the floor and various visual distractions around the landing pad. There are two experiments performed, each with a different background as shown in Fig. 9. Note completely different floor surfaces, where one is reflective the other is matt. Due to the size of the experimentation room, the velocity of the ground vehicle is limited to 1 m/s. The neural network was running on a separate laptop and the image from camera and the velocity command were exchanged wireless.

2) Experimental Performance: The experiment was performed for 5 different speeds of the landing platform in two backgrounds. Each speed setting was repeated 10 times and the results are summarized in Table III. The table shows that the performance drops with increased speed. However, at all speeds the proposed approach has higher success rate than those reported in [11], which used low level states such as the position

information to achieve landing. While two experiments are not comparable, this provides indication that the proposed method can at least match with RL-based methods based on low level states. Note that the two backgrounds tested were never seen during training; instead the neural network learned how to ignore the background and focus on the landing pad only. The experimental video can be viewed here: https://youtu.be/BXyjc6C6TgM, while graphical representation of a sample experimental run is shown in Fig. 10.

VI. Conclusion

This letter presented an end-to-end deep reinforcement learning approach for autonomous landing on a moving platform. The proposed approach is the first attempt at autonomous landing of a UAS on a moving platform using an end-to-end vision-based approach with vertical velocity control. To allow RL working with vision, a state-of-the-art reinforcement learning algorithm, DrQv2, was employed. Additionally, a new reward was proposed, which (i) keeps negative rewards to the minimum, (ii) gives rewards for desired behaviours of descending at the safe speed and (iii) gives rewards for keeping the landing pad within the LOS of the UAS. Through extensive simulations, the performance of the proposed algorithm was verified. It was shown that the proposed reward performs better than other state-of-the-art rewards in vision-based approaches. Finally, domain randomization was used to perform an experiment with CrazyFlie 2.1 vehicle. It was shown that proposed approach performs well in the experimental settings. Future work can focus on using advanced vision techniques to increase the experimental performance by closing the visual gap between the simulated and real images. Another direction for the future work is expanding the scenarios to include more complex outdoors environments with obstacles. Additionally, more complex motion of the landing platform (e.g., nonlinear trajectory and uneven speed) should be added, which would potentially require memory networks.

References

[1] S. Hutchinson, G. Hager, and P. Corke, "A tutorial on visual servo control," *IEEE Trans. Robot. Automat.*, vol. 12, no. 5, pp. 651–670, Oct. 1996.

[2] S. Yang, J. Ying, Y. Lu, and Z. Li, "Precise quadrotor autonomous landing with SRUKF vision perception," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 2196–2201.

[3] C. Chen et al., "Obtaining world coordinate information of UAV in GNSS denied environments," *Sensors*, vol. 20, no. 8, pp. 1–24, 2020.

[4] C. Chen, S. Chen, G. Hu, B. Chen, P. Chen, and K. Su, "An auto-landing strategy based on pan-tilt based visual servoing for unmanned aerial vehicle in GNSS-denied environments," *Aerosp. Sci. Technol.*, vol. 116, 2021, Art. no. 106891.

[5] W. Zhao, H. Liu, and X. Wang, "Robust visual servoing control for quadrotors landing on a moving target," *J. Franklin Inst.*, vol. 358, no. 4, pp. 2301–2319, 2021.

[6] F. Chaumett and S. Hutchinson, "Visual servo control Part I: Basic approaches," *IEEE Robot. Automat. Mag.*, vol. 13, no. 4, pp. 82–90, Dec. 2006.

[7] B. Herisse, F. X. Russotto, T. Hamel, and R. Mahony, "Hovering flight and vertical landing control of a VTOL unmanned aerial vehicle using optical flow," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2008, pp. 801–806.

[8] D. Lee, T. Ryan, and H. J. Kim, "Autonomous landing of a VTOL UAV on a moving platform using image-based visual servoing," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2012, pp. 971–976.

[9] I. Borshchova and S. O'young, "Visual servoing for autonomous landing of a multi-rotor UAS on a moving platform," *J. Unmanned Veh. Syst.*, vol. 5, no. 1, pp. 13–26, 2016.

[10] G. Cho, J. Choi, G. Bae, and H. Oh, "Autonomous ship deck landing of a quadrotor UAV using feed-forward image-based visual servoing," *Aerosp. Sci. Technol.*, vol. 130, 2022, Art. no. 107869.

[11] A. Rodriguez-Ramos, C. Sampedro, H. Bavle, I. G. Moreno, and P. Campoy, "A deep reinforcement learning technique for vision-based autonomous multirotor landing on a moving platform," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 1010–1017.

[12] C. Sampedro, A. Rodriguez-Ramos, I. Gil, L. Mejias, and P. Campoy, "Image-based visual servoing controller for multirotor aerial robots using deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 979–986.

[13] A. Rodriguez-Ramos, C. Sampedro, H. Bavle, P. de la Puente, and P. Campoy, "A deep reinforcement learning strategy for UAV autonomous landing on a moving platform," *J. Intell. Robot. Syst.: Theory Appl.*, vol. 93, pp. 351–366, 2019.

[14] J. Xie, X. Peng, H. Wang, W. Niu, and X. Zheng, "UAV autonomous tracking and landing based on deep reinforcement learning strategy," *Sensors*, vol. 20, pp. 1–17, 2020.

[15] R. Polvara, S. Sharma, J. Wan, A. Manning, and R. Sutton, "Autonomous vehicular landings on the deck of an unmanned surface vehicle using deep reinforcement learning," *Robotica*, vol. 37, pp. 1867–1882, 2019.

[16] R. Polvara et al., "Toward end-to-end control for UAV autonomous landing via deep reinforcement learning," in *Proc. Int. Conf. Unmanned Aircr. Syst.*, 2018, pp. 115–123.

[17] R. Polvara, M. Patacchiola, M. Hanheide, and G. Neumann, "Sim-to-real quadrotor landing via sequential deep q-networks and domain randomization," *Robot.*, vol. 9, pp. 1–20, 2020.

[18] J. Wang, T. Wang, Z. He, W. Cai, and C. Sun, "Towards better generalization in quadrotor landing using deep reinforcement learning," *Appl. Intell.*, vol. 53, pp. 6195–6213, 2022.

[19] Y. Xu, Z. Liu, and X. Wang, "Monocular vision based autonomous landing of quadrotor through deep reinforcement learning," in *Proc. Chin. Control Conf.*, 2018, pp. 10014–10019.

[20] J. E. Kooi and R. Babuška, "Inclined quadrotor landing using deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 2361–2368.

[21] D. Yarats, R. Fergus, A. Lazaric, and L. Pinto, "Mastering visual continuous control: Improved data-augmented reinforcement learning," 2021, *arXiv:2107.09645*.

[22] K. Ciosek, R. Loftin, Q. Vuong, and K. Hofmann, "Better exploration with optimistic actor-critic," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 1787–1798.

[23] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, "Learning to fly–A gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 7512–7519.

[24] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[25] P. Ladosz, L. Weng, M. Kim, and H. Oh, "Exploration in deep reinforcement learning: A survey," *Inf. Fusion*, vol. 85, pp. 1–22, 2022.

[26] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2016.

[27] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[28] E. Coumans and Y. Bai, "Pybullet, a Python module for physics simulation for games, robotics and machine learning," 2016–2021. [Online]. Available: http://pybullet.org