Single Reinforcement Learning Policy for Landing a Drone under Different UGV Velocities and Trajectories

Jose Amendola*
Dept. of Engineering Sciences
University of Agder
Grimstad, Norway
jose.amendola@uia.no

Linga Reddy Cenkeramaddi
Dept. of Information and Communication Technology
University of Agder
Grimstad, Norway
linga.cenkeramaddi@uia.no

Ajit Jha
Dept. of Engineering Sciences
University of Agder
Grimstad, Norway
ajit.jha@uia.no

2023 11th International Conference on Control, Mechatronics and Automation (ICCMA) | 979-8-3503-1568-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCMA59762.2023.10374818

***Abstract*—We propose an algorithm that combines Reinforcement Learning (RL) and a PID cascade control for landing a drone on a moving unmanned ground vehicle (UGV). Unlike other works, where each policy is trained towards one specific task represented by one ground vehicle velocity, here we present an unified policy that enables the drone to land on UGV moving in multiple trajectories with different velocities. This improves the drone's capability and efficiency by reusing the parameters across different scenarios. More specifically, we consider the landing tasks by combining UGV platform linear velocities of 0.1 m/s, 0.2 m/s and 0.3 m/s with angular velocities $-\pi/4$ rad/s, 0 and $+\pi/4$ rad/s. In addition, the velocity set points sent to the platform are considered in the state space, enabling discrimination among situations and successful landing in different conditions with a single policy. The trained policy provides position offset commands to a cascade control, which, in turn, converts it into drone's motor thrusts enabling efficient maneuver. Further, the training occurs with parallel threads collecting experiences under different platform velocities and the actions are given in larger time window, compatible with real world applications where latency exists between sensor processing and data transmissions. Finally, the simulated experiments reveals convergence and demonstrate efficient landings for different conditions. The usage of an underlying cascade control exempts the policy from learning to stabilize the drone locally.**

***Index Terms*—Drone, UGV, Dynamic Landing, Reinforcement Learning**

I. Introduction

As a growing number of applications make use of autonomous navigation in different transport modes, it is expected that different autonomous platforms interact with each other over different means, such as water, air, and ground. As aerial vehicles, drones provide a privileged view of environments, at the expense of limited payload and power autonomy. On the other hand, unmanned ground vehicles (UGVs) can carry higher payloads and battery capacity [1] [2]. A typical interaction between those two types of platforms that leverages their complementarity is the landing of drones on moving UGVs [3]. This interaction leverages missions that benefit from aerial affordances backed up by higher autonomy and robustness with ground support.

Autonomous landing on a moving platform poses some technical challenges. It requires reliable reading on the platform since it is not stationary. From the data acquisition perspective, estimating the relative position of the platform in real-time with high accuracy is the key to a successful landing. Computer vision algorithms can be sensitive to lightning and visual variations and might not be sufficiently fast to attend to precise control requirements. Transmitting data between the ground platform and the drone could also suffer from interference and channel limitations. From the control perspective, dealing with delays in data acquisition requires advanced control techniques and a daunting tuning of parameters.

Some approaches employ purely Reinforcement Learning in its end-to-end form to the landing task, where images are directly mapped into actuation commands for the drone. While RL allows the emergence of complex behaviors, it can be computationally expensive and presents limited generalization among scenarios.

In this paper, we leverage the strengths of the RL algorithms for the autonomous landing of drones on a moving UGV platform, complementing it with the adoption of simplified state representation and the simplicity of PID control. The actions provided by the RL policy give drone position offsets in a time window compatible with the real world applications that include estimation and/or transmission of parameters between UGV and drone, while the simplified controller guarantees stability and controls the drone thrusts in shorter steps to pursue those offsets. We included simplified context variables in the state representation so the algorithm learns to land under different platform moving conditions, improving the generalization of the method.

This paper is structured as follows: Sec. II presents related works. Sec. III briefly presents the principles and algorithm

This work is supported by SFI Offshore Mechatronics project: 237896, the Indo-Norwegian Collaboration in Autonomous Cyber-Physical Systems (INCAPS) project: 287918.

used in this paper. Sec. IV describes the proposal in detail; Sec. V presents the experiments and results. Finally, Sec. VI concludes the paper.

II. Related Work

Typically, the drone landing task is approached as a control task in the literature. In this case, there is the assumption of knowledge of the system dynamics (at least partially) and the data availability is taken for granted [4]. In those approaches, a simpler controller is used where limitations are accounted for or more robust controllers are employed at the expanse of more parameter tuning [5].

Reinforcement Learning (RL) is a machine learning approach that has been largely explored in autonomous navigation tasks of many sorts [6] [7]. It does not require prior knowledge of system dynamics and can integrate behaviors such as collision avoidance, target tracking, and goal navigation. In aerial vehicles domain, for example, using RL to map action directly from images suffers from poor generalization and suffers from sim-to-real gap [8].

Most of the literature that uses RL for landing a drone on a static platform maps actions from simplified vectors with relative position and velocity. The tasks are simplified by tracking the target platform on the horizontal plane only. The vertical descent is detached and occurs based on a pre-defined rule [9] [10].

When it comes to landing on moving platforms, especially using RL, there are fewer works in the literature. It is possible to see more complex state representations, including, for example, the yaw value for the drone [11]. Some of them further decouple vertical descent with heuristic rules [12]. There are also works that use RL in a hierarchical fashion, switching among different landing phases for further downstream control [13]. In most cases, the landing is considered successful by touching a marker with angular positions within a range.

Moving away from the trend in the literature, our proposal uses a single RL policy for the drone's position offsets in all three dimensions (x,y,z), incorporating its PID controller output and further prolonging the drone touching to consider the landing as successful. The state space is augmented with variables that allow the policy to generalize the landing for different RL tasks, represented by different ground vehicle velocities and trajectories. This arrangement exempts the RL algorithm from the complexity of learning drone stabilization. We also employ a reward function with a shaping term, aiming to optimize convergence.

III. Background

RL involves constructing an agent's behavior using actions and rewards [14]–[17]. In this process, the agent observes a state, takes an action leading to a state transition, and receives a reward signal, modeled as a Markov Decision Process (MDP). MDPs adhere to the Markov property where transitions to new states depend solely on the current state and the action taken.

An MDP is characterized by state set S, action set A, transition model $T(s, a, s')$ specifying probabilities of transitioning from state s to s' after action a, reward function $r \in R : S \times A \times S \rightarrow \mathbb{R}$, and discount factor γ weighing future rewards. A policy $\pi : S \rightarrow A$ maps states to actions, and RL aims to optimize this policy for maximum reward.

Proximal Policy Optimization (PPO) [18] is a widely-used RL algorithm that optimizes policies for sequential decision-making and control tasks. It gained popularity due to its simplicity and added stability to training.

PPO balances policy enhancement while keeping new policies near the old ones to ensure stability. This involves introducing a surrogate objective function approximating policy improvement and capping policy updates. By restraining policy updates to a range near the old policy, PPO prevents the policy shifts from destabilization.

To confine policy updates, PPO employs a clip function constraining the ratio of new and old policies within a range specified by $(1 - \epsilon)$ and $(1 + \epsilon)$:

$$L_{\text{clip}}(\theta) = \mathbb{E}\left[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\pi_{\theta_{\text{old}}}}(s,a), \rho A^{\pi_{\theta_{\text{old}}}}(s,a)\right)\right] \tag{1}$$

Here, $A^{\pi_{\theta_{\text{old}}}}(s,a)$ is the advantage function quantifying the action's advantage compared to the old policy's expected value. ρ is the ratio between the probability of taking action a from state s in the current policy and in the older from previous step respectively. The function employs a clip function constraining the ratio of new and old policies within a range specified by $(1 - \epsilon)$ and $(1 + \epsilon)$. By optimizing this clipped objective function using gradient ascent, PPO attains stable policy updates, striking a balance between exploration and exploitation.

IV. Method

We propose an RL-based framework for landing a drone on a UGV that moves with different trajectories. We assume an underlying cascade control for its position and that the trained agent (drone) can give position offset in spaced intervals. This enables incorporating behaviors on a higher level, and at the same time, the policy can be more easily adapted for

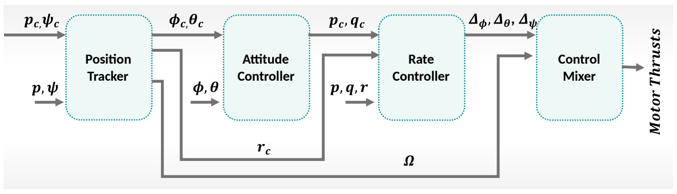

Fig. 1. Cascade drone position controller: The position tracker modules handle new position and yaw inputs, utilizing the altitude controller for vertical positioning and the horizontal controller for movement in the X-Y plane. The yaw controller determines the required yaw rate. The attitude controller module stabilizes the drone based on angular position data from the horizontal controller. It calculates pitch and roll errors and estimates angular velocities. The rate controller module computes torque variations necessary to achieve the angular velocities dictated by the attitude controller. The control mixer module allocates thrust to the motors considering altitude requirements and torque variations.

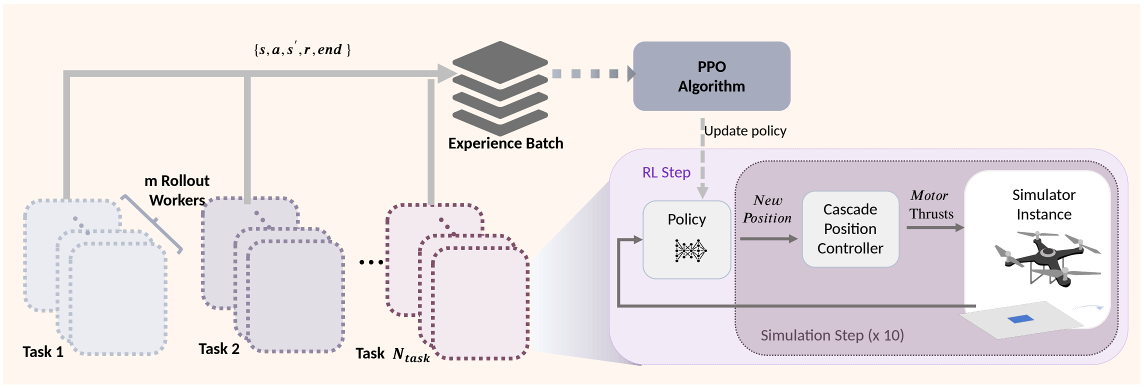

Fig. 2. General Diagram: Different rollout workers run episodes with environment instances. Each rollout is assigned a desired platform velocity. At each RL step, they send an experience tuple to the PPO algorithm batch. Once the batch size limit is achieved, the PPO algorithm updates the policy network parameters for all the rollout workers. In each RL step, the policy network receives the state (drone-platform relative position and velocity), augmented by the platform velocity fixed setpoints, and gives position offsets as actions. The cascade controller gives thrust commands to the drone motors to pursue the offset. The step involving simulation integration and control loop occurs 10 times inside the RL loop.

different drone models and/or PID control parameters without the burden of learning how to stabilize the drone.

a) Drone controller: The drone controller adopted considers a linearized and decoupled model for the drone. It is based on a cascade of four modules, as proposed by [19]. The first module is a position tracker that receives the new position p_c to pursue and the new yaw value ψ_c. In this module, the altitude controller provides the necessary thrust Ω to achieve the new vertical position, while the horizontal controller calculates the necessary roll and pitch angles (ϕ_c and θ_c) to move the drone in the X-Y plane. The yaw controller estimates directly the yaw rate r_c needed to achieve the new yaw value. In this work, we assume the drone will always land pursuing a yaw value equal to 0 degrees. The next module is the attitude controller, which stabilizes the drone at the desired angular position provided by the horizontal controller from the position tracker module. It uses the error of pitch and roll angles and estimates the angular velocities p_c and q_c. Those values are sent to the rate controller module along with the r_c calculated in the first module. Here, the goal is to calculate the total torque variations Δ_ϕ, Δ_θ and Δ_ψ from the equilibrium point of motors required to create the angular velocities dictated by the previous module. Finally, the control mixer module distributes the thrust to the motors based on the drone configuration and design. It considers the thrust needed for altitude Ω calculated by the position tracker and the variations calculated by the rate controller (Δ_ϕ, Δ_θ, Δ_ψ). The block diagram of the cascade controller is depicted in Fig.1.

b) Reinforcement learning: Training RL policy to stabilize drones with different dynamics is challenging. In this work, the policy learns to navigate assuming an underlying response from the controller. Similar responses can be achieved in many combinations of control parameters and model dynamics. The policy offers position offsets as actions to lead the drone toward a successful landing. This type of command is supported by many drone firmware arrangements and there is a clearer separation between path planning (giving positions where to go) and control. The interval between commands also allows most sensing and data transmission to occur without compromising drone stability.

Algorithm 1 Episode Collection

```
Input: Current policy $\pi$, Environment, Task parameters $||_m v_d||$, $_m\omega_d$ assigned for the worker
Output: Sequence of experience tuples
1:  $s \leftarrow$ Reset Environment
2:  Assign $||_m v_d||$, $_m\omega_d$ as platform velocity setpoint
3:  while Episode not ended do
4:      $p^{sp} \leftarrow$ Increment offsets sampled from policy
5:      for 10 Simulation Steps do
6:          Motor Thrusts $\leftarrow$ Cascade Controller
7:          Step Simulation
8:      end for
9:      $s' \leftarrow$ Calculate New State
10:     if Touched Inner Landing Area then
11:         Wait $t_{\text{land}}$ s
12:         if Still touching Inner Landing Area then
13:             Landed $\leftarrow$ true
14:         end if
15:     end if
16:     if Touched Anything Else then
17:         Collided $\leftarrow$ true
18:     end if
19:     if Landed or $T_{max}$ reached or Collided or Drone Exceeded Boundaries then
20:         End Episode
21:     end if
22:     Calculate Reward
23:     Send tuple $< s, a, s', r, end >$ to PPO batch
24:     $s \leftarrow$s'
25: end while
```

We define the interval between actions obtained by the policy as Δt_{policy} and the time step used for numerical integration of the simulation as Δt_{sim}. Drone position is defined by $_d p_t = \{_d p_t^x, _d p_t^y, _d p_t^z\}$, its velocity by $_d v_t = \{_d v_t^x, _d v_t^y, _d v_t^z\}$ and the action output calculated by the policy network as $a_t = \{x_t^{offset}, y_t^{offset}, z_t^{offset}\}$. The position set-point to be sent to the cascade controller is $p_t^{sp} = \{_d p_t^x + x_t^{offset}, _d p_t^y + y_t^{offset}, _d p_t^z + z_t^{offset}\}$. The value p_t^{sp} used by the controller

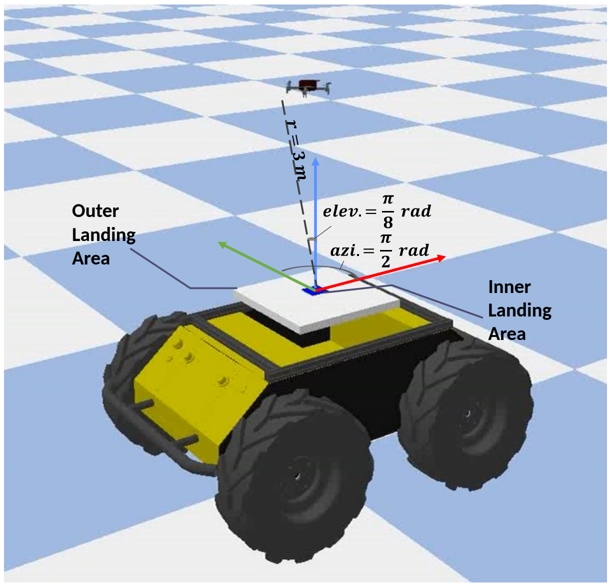

Fig. 3. Simulation Environment: Above the Husky UGV (yellow), the outer landing area is in white with the inner landing area in blue in the center. The initial position of the drone is defined in spherical coordinates from the landing area center. The radius is 3 m , elevation angle of $\frac{\pi}{8}$ and azimuth $\frac{\pi}{2}$.

remains unchanged from t until $t+\Delta t_{policy}$. In this work, we adopted $\Delta t_{sim} = 0.02$ s and $\Delta t_{policy} = 10\Delta t_{sim} = 0.2$ s.

During training, the action values are sampled from a normal distribution with parameters - mean (μ) and standard deviation (σ) vectors (one value for each action dimension) obtained by the policy network trained with PPO. At test time, the (σ) values are set to 0 and the policy becomes deterministic. The offset given as actions were limited to values between -0.2 m and $+0.2$ m. The overall pipeline is illustrated in Fig. 2. At every training step, the PPO Algorithm receives a batch of size B, experience tuples ($< s, a, s', r, end >$) for each state transition, with end flag representing whether the transition is ending an episode. Parallel roll-out workers (parallel threads of computation) have instances of the policy, and the environment, and keep running episodes and sending the experience tuples to the batch.

Different UGV velocities and trajectories result in different state transitions perceived by the drone policy during the landing process. According to RL formality, each UGV velocity implies a different RL task. Given that the policy must train in different UGV velocities, we instantiate the rollout workers to run episodes in a given UGV velocity condition. To balance the experience tuples sent to the batch, it is desirable that there are equal numbers of workers for each task. We parameterize each task by two parameters: The UGV linear velocity modulus $||_m v_d||$ and its angular velocity $_m\omega_d$. Using a simplified kinematic model for a four-wheeled skid steering platform, the desired wheel rotations are obtained and passed by the UGV controllers. Let N_{tasks} be the total number of value combinations, we distribute the conditions equally, summing up to $N_{tasks}m$ rollout workers, with m an integer multiplier value. Each episode rollout is described in Alg. 1.

Considering the position and velocity of the landing target

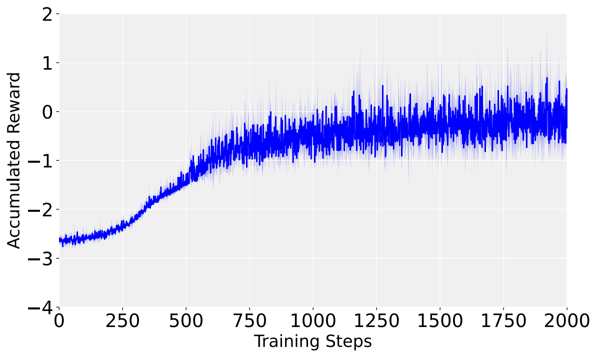

Fig. 4. Training curves: Accumulated reward per episode over training steps. For each case the curve is plotted as the mean of 3 runs filled by a region representing standard deviation.

at the mobile platform (UGV) $_m p_t = \{_m p_t^x, {_m p_t^y}, {_m p_t^z}\}$ and $_m v_t = \{_m v_t^x, {_m v_t^y}, {_m v_t^z}\}$, the state space adopted s_t considers the position and velocity of the drone relative to the mobile platform $_r p_t = {_m p_t} - {_d p_t}$ and $_r v_t = {_m v_t} - {_d v_t}$. It also includes two variables. The moving platform desired parameters $||_m v_d||$ and $_m\omega_d$. Even though this information is already implicitly accounted for in the relative velocity, the absolute platform values work to discriminate situations with similar relative velocity, but with different needed behaviors to achieve a successful landing. Different UGV velocities translate into different RL tasks and that might require the drone policy to give a distinct sequence of actions, especially if there is a curve performed by the UGV – in which case the drone might anticipate a more straight trajectory in order to minimize the total time to landing. Notably, those two last variables added will remain constant for the whole episode and are essential to five multi-tasking ability to the policy. The real UGV velocity will fluctuate to achieve that set point, but adopting those changing values would add unnecessary complexity. That leads to

$$s_t = \{_r p_t^x, {_r p_t^y}, {_r p_t^z}, {_r v_t^x}, {_r v_t^y}, {_r v_t^z}, ||_m v_d||, {_m\omega_d}\} \quad (2)$$

The reward function r_t is given by

$$r_t = \phi(t) + F_{shaping}(t) \quad (3)$$

where $\phi(t)$ is a potential field and $F_{shaping}(t) = \gamma\phi(t) - \phi(t-1)$ is a shaping term for accelerating convergence. The field function considers relative distance and velocity. Additionally, a constant value R_{dock} is added when the drone lands successfully, touching the inner landing area. The formula is given by:

$$\begin{aligned}\phi(t) = -k(f(_r p_t^x) + f(_r p_t^y) + f(_r p_t^z) + \\ f(_r v_t^x) + f(_r v_t^y) + f(_r v_t^z)) \\ +R_{dock}\end{aligned} \quad (4)$$

with

$$f(x) = \begin{cases} |x| & \text{if } |x| < 1 \\ x^2 & \text{if } |x| \geq 1 \end{cases} \quad (5)$$

The modulus is used instead of a quadratic function for the fact that it yields higher values for very small relative positions and velocities – which is essential for actions where

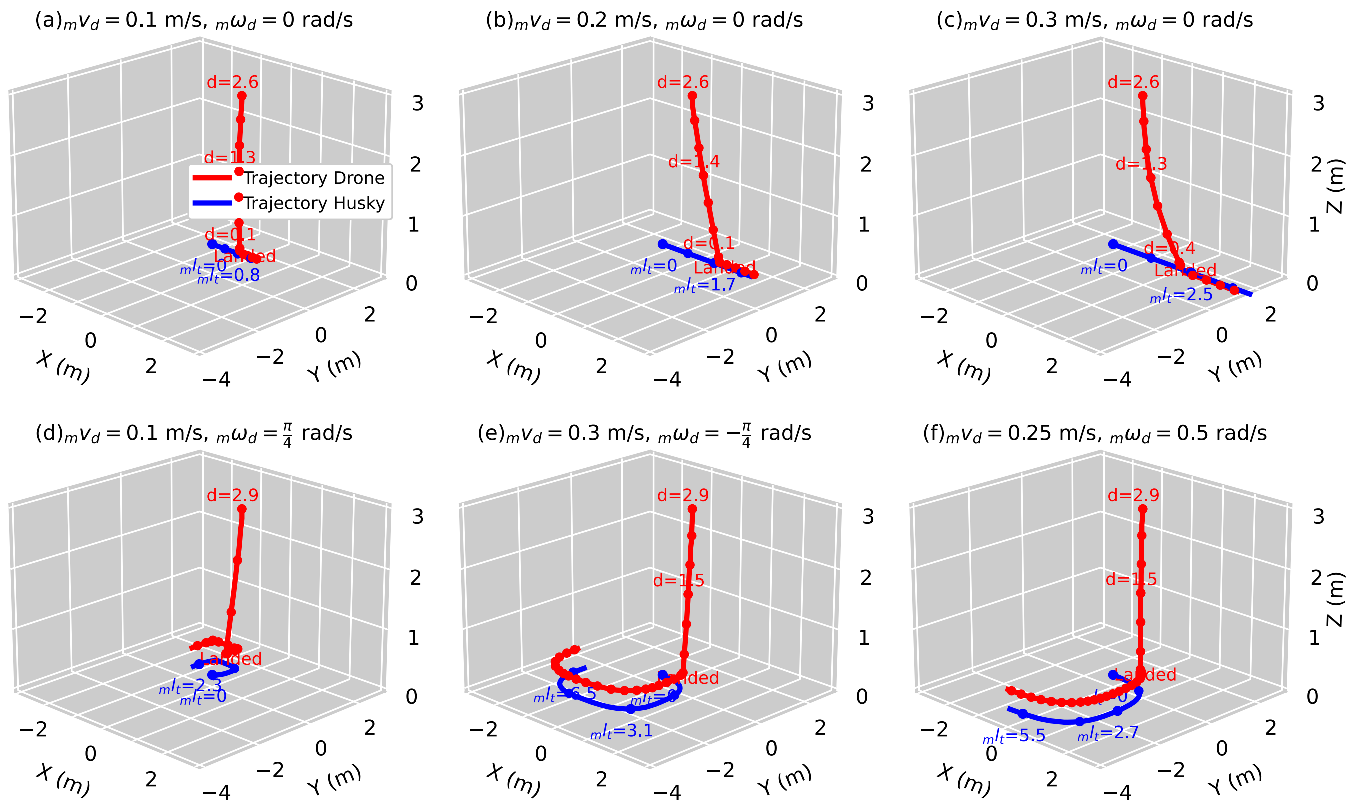

Fig. 5. Trajectories of both the drone and the vehicle(Husky) for different landing tasks (different husky velocities). The $_ml_t$ values show how far the husky UGV moved and is marked in red circles every 15 RL steps. Values d indicate the distance between the drone and the platform, trajectory is marked in blue circles at every 5 RL steps. Tasks (a-e) were sampled from the tasks used in training. Task (f) parameterized by a husky trajectory unseen during training.

the drone is very close to the landing area. The values are set to $k = 0.001$ and $R_{dock} = 30$ if the drone successfully lands on the platform, $R_{dock} = 0$ otherwise.

Each episode ends under the following conditions: If the drone successfully lands (remains touching the blue area in Fig. 3 for two successive RL steps); If it collides with the floor or with the platform (not touching the blue area); If it flies outside a predefined boundary $d_{boundary}$; Or if the number of RL steps exceeds $T_{max} = 400$.

To ensure that the drone lands safely at the landing area and to avoid abrupt touching with bouncing or fallouts afterward, we consider only successful landing if the drone remains in contact with the inner landing area for the period of $t_{land} = 2\Delta t_{policy}$. Once it reaches that state, the drone propellers are automatically turned off.

The policy network architecture adopted was a fully connected network with 2 hidden layers of 256 neurons each. Their activation function of hidden layers was ReLu.

V. Experiments

We performed simulations using the PyBullet engine. The drone model implementation adopted followed [20] and the model adopted is inspired by Bitcraze's Crazyflie 2.x nano-quadrotor. As it is a mobile platform, we adopted the model of the ground vehicle Clearpath Husky and added both outer and inner landing areas, as seen in Fig 3. The Husky vehicle is positioned at the origin with its heading at the x-axis.

The husky desired velocity parameters were set to a combination of the speed magnitude set $S_v = \{0.1, 0.2, 0.3\}$ m/s and the angular velocity set $S_\omega = \{-\frac{\pi}{4}, 0, \frac{\pi}{4}\}$ rad/s, resulting in the condition set $S_v \times S_\omega$ with $N_{tasks} = 9$ velocity conditions. The multiplier for rollout workers was set to $m = 3$, meaning that a total of 27 rollout workers ran episodes, with every condition being repeated in 3 worker threads.

The PID parameters adopted in each of the modules of the cascade controllers followed the implementation for the Craziflie model in [19] [20].

The optimization algorithm used was stochastic gradient descent optimization (SGD) with learning rate $l_r = 5 \times 10^{-6}$. PPO was implemented through the RLLIB framework [21], where each batch size was $B = 4096$. The RL discount factor was set to $\gamma = 0.9$ and PPO hyperparameter $\epsilon = 0.3$. The training occurred for 12000 iterations.

For improving robustness, the initial drone position was set with a slight random deviation. Following a spherical coordinate system with Husky as the center, the initial radius was set to 3 m , the elevation of $\pi/8$ rad and the azimuth of $\pi/2$ (as described in Fig. 3).

The training curve is shown in Fig. 4. The values represent the accumulated reward over episodes along the training. It is possible to see that for all three cases, there is convergence within 2000 algorithm steps.

The trajectories obtained for the policy is shown in Fig. 5 (a-f). The first landings occurred when Husky moved in a straight line with velocity setpoints of 0.1 m/s (a), 0.2 m/s (b), and 0.3 m/s (c). As expected, the higher the husky velocity, the longer it takes for the landing to occur. The drone trajectory indicates an efficient approximation in all cases. For curved husky trajectories, the policy was able to anticipate the curves and reach the UGV proximity in a nearly straight line, landing successfully while the husky executes a curved trajectory. The gap seen between husky and drone trajectories in cases (a), (b), and (c) after landing occurs due to the landing criteria, which allow touching with the inner landing area. The circle markings in the drone trajectory shown in the figure occur at every 5 RL steps and show low-speed variation until the drone is really close to the landing stage. That demonstrates the efficiency of the landing and also that the policy learned to avoid risky accelerations and drop-offs on top of the UGV. This was enabled by the strategy of waiting two steps until the landing was successful during training.

It is expected that the landing in curved UGV trajectories might not be so aligned. That occurs especially because of the value fluctuations while trying to attain the velocity set point summed up with imprecision due to skidding. Notably, the linear angular velocities set to the UGV in case (f) (0.25 m/s and 0.5 rad/s, respectively) were not used during training. That demonstrates the ability of the policy to generalize its ability within the range of the velocities used during training.

VI. Discussion and Conclusion

We presented an algorithm based on Reinforcement Learning for autonomously landing a drone on a moving platform with different velocity conditions. We employed a cascade controller to receive limited position offset commands, so the policy could give actions in a more spaced interval. By providing position offsets as actions, the proposal allowed the policy to be compatible with most drone navigation systems. The results demonstrated that, despite spaced policy time intervals and limited controllability, the policy could successfully learn to land the drone under different velocities of the UGV. The time window between actions is large enough to allow data transmission and sensor processing pipelines.

The usage of variables referring to UGV velocity setpoints enabled the generalization of the policy across different UGV trajectories. At the same, time they simplified the state-exploration by avoiding the real-time velocity varying over the episodes. The employment of threads running under different tasks also led to more training convergence.

References

[1] X. Xu, R. Zhang, and Y. Qian, "Location-based hybrid precoding schemes and qos-aware power allocation for radar-aided uav–ugv cooperative systems," *IEEE Access*, vol. 10, pp. 50 947–50 958, 2022.

[2] T. Miki, P. Khrapchenkov, and K. Hori, "Uav/ugv autonomous cooperation: Uav assists ugv to climb a cliff by attaching a tether," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8041–8047.

[3] G. Quaglia, P. Cavallone, and C. Visconte, "Agri_q: agriculture ugv for monitoring and drone landing," in *Mechanism Design for Robotics: Proceedings of the 4th IFToMM Symposium on Mechanism Design for Robotics*. Springer, 2019, pp. 413–423.

[4] L. Qian, S. Graham, and H. H.-T. Liu, "Guidance and control law design for a slung payload in autonomous landing: A drone delivery case study," *IEEE/ASME Transactions on Mechatronics*, vol. 25, no. 4, pp. 1773–1782, 2020.

[5] N. Dalwadi, D. Deb, and J. J. Rath, "Biplane trajectory tracking using hybrid controller based on backstepping and integral terminal sliding mode control," *Drones*, vol. 6, no. 3, p. 58, 2022.

[6] J. Amendola, L. S. Miura, A. H. R. Costa, F. G. Cozman, and E. A. Tannuri, "Navigation in restricted channels under environmental conditions: Fast-time simulation by asynchronous deep reinforcement learning," *IEEE Access*, vol. 8, pp. 149 199–149 213, 2020.

[7] A. Dayal, L. R. Cenkeramaddi, and A. Jha, "Reward criteria impact on the performance of reinforcement learning agent for autonomous navigation," *Applied Soft Computing*, vol. 126, p. 109241, 2022.

[8] A. Devo, J. Mao, G. Costante, and G. Loianno, "Autonomous single-image drone exploration with deep reinforcement learning and mixed reality," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5031–5038, 2022.

[9] L. Bartolomei, Y. Kompis, L. Teixeira, and M. Chli, "Autonomous emergency landing for multicopters using deep reinforcement learning," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 3392–3399.

[10] M. B. Vankadari, K. Das, C. Shinde, and S. Kumar, "A Reinforcement Learning Approach for Autonomous Control and Landing of a Quadrotor," *2018 International Conference on Unmanned Aircraft Systems, ICUAS 2018*, pp. 676–683, aug 2018.

[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," *IEEE International Conference on Intelligent Robots and Systems*, pp. 1010–1017, dec 2018.

[12] J. Xie, X. Peng, H. Wang, W. Niu, and X. Zheng, "UAV Autonomous Tracking and Landing Based on Deep Reinforcement Learning Strategy," *Sensors 2020, Vol. 20*, vol. 20, no. 19, p. 5630, oct 2020.

[13] R. Polvara, M. Patacchiola, S. Sharma, J. Wan, A. Manning, R. Sutton, and A. Cangelosi, "Toward End-to-End Control for UAV Autonomous Landing via Deep Reinforcement Learning," *2018 International Conference on Unmanned Aircraft Systems, ICUAS 2018*, pp. 115–123, aug 2018.

[14] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[15] V. Dutt, "Explaining human behavior in dynamic tasks through reinforcement learning," *Journal of Advances in Information Technology*, vol. 2, no. 3, pp. 177–188, 2011.

[16] N. Muslim, S. Islam, and J.-C. Grégoire, "Reinforcement learning based offloading framework for computation service in the edge cloud and core cloud," *Journal of Advances in Information Technology Vol*, vol. 13, no. 2, 2022.

[17] T. S. Hlalele, Y. Sun, and Z. Wang, "Intelligent fault detection based on reinforcement learning technique on distribution networks," *Journal of Advances in Information Technology*, vol. 14, no. 3, 2023.

[18] B. Liu, Q. Cai, Z. Yang, and Z. Wang, "Neural trust region/proximal policy optimization attains globally optimal policy," *Advances in neural information processing systems*, vol. 32, 2019.

[19] C. Luis and J. L. Ny, "Design of a trajectory tracking controller for a nanoquadcopter," *arXiv preprint arXiv:1608.05786*, 2016.

[20] 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 *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 7512–7519.

[21] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "Rllib: Abstractions for distributed reinforcement learning," in *International conference on machine learning*. PMLR, 2018, pp. 3053–3062.