Closing the Sim-to-Real Gap: Enhancing Autonomous Precision Landing of UAVs with Detection-Informed Deep Reinforcement Learning

Charalambos Soteriou[†][0009−0004−8502−9976], Christos Kyrkou[†][0000−0002−7926−7642], and Panayiotis S. Kolios[†,‡][0000−0003−3981−993X]

[†]KIOS Research and Innovation Center of Excellence,
[‡]Computer Science Department,
University of Cyprus, Cyprus, University of Cyprus, Cyprus,
{soteriou.charalambos,kyrkou.christos,kolios.panayiotis}@ucy.ac.cy
https://www.kios.ucy.ac.cy/

Abstract. Autonomous precision landing of a UAV is a challenging task relying on simultaneous target localization and control. It can not be performed using GPS coordinates, due to limitations in the accuracy of the technology. Control policies are fine-tuned in simulation before deployment but most simulators are equipped with low-quality graphics, posing a challenge when utilizing vision based algorithms, intended to be implemented in the real world. In this paper we showcase a joint computer vision and reinforcement learning approach, in the photo-realistic simulator AirSim, to reduce the sim-to-real gap. Localization is performed using the Yolov8 object detector and control using the PPO and LSTM-PPO algorithms. We achieve a 94.25% and 94.86% success rate, over 1391 and 817 landings at three different simulated environments, respectively.

Keywords: Autonomous UAV landing · Reinforcement Learning · Photo-realistic simulation · Object Detection

1 Introduction

Rotor propelled UAV's such as quadrotors, have been proven to be extremely useful in a plethora of applications such as surveying, mapping and monitoring [15]. However they suffer from battery capacity restrictions, severely limiting their flight time and by extend utility [30]. This limitation can be overcome by using a portable charging station to recharge the drone and allow it to continue its mission without having to return to its operator. Routine autonomous missions can be carried out based on GPS coordinates because they don't require precision less than 3m, which is the limit of GPS technology [6]. Landing on a smaller surface however, cannot be carried out by relying on GPS accuracy. The charging station would need to be identified and localized while simultaneously a control policy dynamically maneuvers the drone on the target.

A control policy either explicitly models the dynamics of the system if the equations of motion are known or approximates the dynamics for more complex systems. In most cases directly modelling the dynamics is not possible and we have to resort to approximations. A common control approach is to use a Proportional Interval Derivative (PID) controller which is a control loop mechanism that continuously calculates the difference between a desired value and a measured process variable and correlates it through a proportional term, it's derivative and it's integral [1]. Reinforcement Learning has also been proven to be a promising control approach and is widely used in robotics [12]. In RL a policy is learned while the agent interacts with its environment and receives a reward signal for actions that help it achieve a predefined objective. At each timestep the agent performs an action sampled from the policy and receives a reward signal when the episode terminates. At the end of each episode the agents state is reset. The optimal policy is the one that maximizes the cumulative reward over the time horizon.

A major challenge in deploying control policies and particularly RL, is that the policies are unstable during training and need to first be tuned in simulation to avoid damaging the hardware. An ongoing challenge is therefore an effective transition of the in-simulation-trained policy to real implementation, also known as sim-to-real gap. To tackle this problem, various simulators have been proposed that provide accurate physics models of robotic arms (e.g. PyBullet), multi-joined-robots (e.g. MuJoCo) or vehicles (e.g. Gazebo). Although these simulators accurately approximate the dynamics of these systems, they lack photo-realistic graphics, restricting the implementation of computer vision tasks, in our case that of target localization.

Object detection is a task in computer vision where an object is identified and localized in an image. A popular approach on visually identifying the position of a target is through the use of fiducial markers, such as Quick Response (QR) [29] and Aruco Markers [9]. Markers offer the capability to use standardized algorithms that translate observed patterns to distance measurements and out-of-the-box implementation foregoing fine-tuning. This approach however, can only be implemented on targets that have a marker attached to them and it reduces real world applications [17].

In contrast, deep learning computer vision algorithms can learn to classify and localize an object based on image input and are implemented in many real-world applications. You Only Look Once (YOLO) algorithms are object detectors that perform detection and localization in a single stage [28]. As opposed to other approaches that use proposal regions to focus on certain parts of the image, YOLO splits the image into a grid cell and searches each cell for objects at multiple scales by varying the stride and window size. This allows YOLO to perform real-time inference speeds, faster than method based on proposal regions, also known as two stage detectors [7].

Despite implementation challenges of computer vision approach such as iterative manual labelling of data and fine tuning, it makes no assumptions for the presence of a marker. Furthermore, computer vision approaches, based on convolutional neural networks, are considered state-of-the-art in real-time object detection and can be used to localize a target from a high altitude, on a live video stream, even on computationally restricted devices.

The existing autonomous landing approaches are trained in low-graphics simulators and use specialized markers to perform localization, restricting generalization when being implemented in the real world. In this paper, we shrink the sim-to-real gap by;

– Implementing a novel approach to autonomous landing that mitigates the sim-to-real gap by leveraging a Yolov8 detection algorithm within a photo-realistic simulator, eliminating the need for specialized markers.
– Enhancing robustness by pre-training and validating the object detector using real-world images.
– Introducing a modular and detection-agnostic methodology, facilitating easy generalization across various scenarios.

Our findings showcase that it is possible to perform precision landing with an accuracy exceeding 94% in a photo-realistic simulator based on the Yolov8 object detector and an RL control policy such as PPO or LSTM-PPO. The policy is trained to centralize the target, regardless of detection id, dimensions or image resolution, allowing for easy generalization to other tasks or new targets.

2 Related Work

An overview of various physics simulators that are used to train robotic agents are presented in [5]. More specifically, it gives an overview of simulation software used in the aerial robotics domain. The most popular simulators for aerial robotics are; Gazebo [13], Webots [16], AirSim [25], jMAVSim and Flightmare [27]. Gazebo is described to have limited rendering capabilities compared to Unity and Unreal Engine. AirSim's resource-intensiveness is it's major drawback compared to other simulators [13]. Flightmare combines a flexible physics engine with the Unity rendering engine, creating a powerful simulator and seems to be suited for deep reinforcement learning applications [27]. jMavSim [2] supports only basic sensing and rendering and has been discontinued. Finally, Webots [16] can support custom physics engine and integrate data from Open-Street Map to create more realistic environments.

In their work [18] the authors introduced, PyBullet-Drones, a multi-agent quadcopter simulator based on the PyBullet Physics engine that emphasises the interface between the simulator and multi-agent RL frameworks. In table 1 they provide a thorough comparison of current aerial robotics simulators such as AirSim, Flightmare and RotorS extension, CrazyS and their work. PyBullet-Drones offers high parallelisation, steppable physics and it is very lightweight but these feature come at the cost of rendering quality which is low. In comparison a quadrotor simulator based on ROS and Gazebo, RotorS [8], does not come with ready-to-use RL interfaces and it's dependency on Gazebo makes it a subpar choice for vision-based learning applications. CrazyS [26] is an extension of RotorS that is specifically targeted to the Bitcraze Crazyflie nano quadcopter, but still suffers from the same limitations as RotorS. Another notable simulator, ETH's Flightmare [27] build on top of the Unity game engine, provides photo-realistic rendering and very fast, highly parallelised dynamics. It also implements OpenAI Gym's API to easily integrate single agent RL workflow but does not support vision-based observations. Finally, AirSim [25] built on top of Unreal Engine 4, is a very popular simulator and enjoys photo-realistic rendering but suffers from elevated computational requirements.

Aerial robotic simulators, in conjunction with Reinforcement Learning control policies, are utilized to train policies in various UAV related tasks. In [10] low-level control of a quadrotor was achieved, mapping states directly to actuator commands, to stabilize a drone after being thrown mid-air at various orientations and speeds. The control policy is trained on fixed length trajectories with inputs being the 3D-rotation matrix, position, angular and linear velocity vectors. For this work the policy was trained in a simulation software developed by the authors, named Robotic Artificial Intelligence (R.A.I.), and then tested in a real environment. It employs a deterministic on-policy algorithm, using zero-bias, zero-variance samples. A value and a policy network are trained, achieving a 96 % success rate.

In [19] a UAV framework is proposed to locate a missing human after a natural disaster using an approximated RL policy to navigate to the target. The state space is defined as the relative distance from the UAV to the nearest target and the distances to the nearest obstacles in each direction; north, south, east and west. The quadrotor navigation is achieved utilizing a PID in conjunction with an approximated Q-learning algorithm to accelerate convergence. The policy is trained in a MATLAB simulation, taking the vanilla Q-learning algorithm 160 episodes to find the optimal course trajectory while the function approximated policy achieved that in 75 episodes. Real-world implementation is carried out on Parrot AR Drone 2.0, using as observations the Motion Capture System from Motion Analysis to provide the relative position with respect to the target and obstacles.

The authors of [3] propose a novel method for learning robust visuomotor policies, for real-world deployment, which can be trained purely with simulated data. By taking cross-model observations in the form of raw camera data and the drone's relative orientation compared to a gate. The observations are represented in a low-dimensional embedding using variational-auto-encoders (VAE) and boasts an improvement in performance compared to other end-to-end control policies or purely supervised feature extractors. To perform these experiments they first generate 300K pairs of 64x64 images along with the relative ground-truth relative gate poses using AirSim [25].

Narrowing our scope, papers such as [21] and [20] achieve autonomous landing of a quadrotor on a marker using a hierarchy of DQN networks. Initially the quadrotor approaches the landing area and when the drone enters the landing zone a different network is triggered to perform a descending maneuver. The networks take as input four stacked, gray-scale images of 84×84 pixel resolution. The observations are processed by the Double-DQN and produce discrete direction outputs. The policy is trained in the Gazebo simulator and utilized domain randomization to better generalize to real scenarios.

The authors of [23] deploy a deep reinforcement learning policy to achieve landing of a quadrotor on a moving platform. The policy is trained using the RotorS UAV simulator build on Gazebo. The moving platform target has dimensions of 1.2m×1.2m and the DDPG algorithm is trained using as inputs the ground truth relative positions, obtained from fiducial markers located on the platform. Successful landing attempts range from 65.45% when the platform is moving at a maximum velocity of 1.2 m/s, to 90.21% when the platform is moving at a maximum velocity of 0.4 m/s.

In their paper [14] the authors achieve landing of a quadrotor on an inclined platform using deep reinforcement learning. A PPO algorithm is trained in two custom Gym-based environments using the roll and pitch angles, as well as the position and velocity vectors, as observations. The authors then deploy a sparse rewards and a tailored curriculum learning approach to achieve an in simulation success rate of 93.3%. They directly transfer the learned policy from simulation to reality, without techniques such as domain randomization, achieving a performance of 86.7%.

Current aerial robotic approaches are using lightweight, but low graphics-quality simulators to perform tasks such navigation, control, localization and more. This in-turn limits the utilization of computer vision based algorithms. More specifically for the autonomous landing task, it creates the necessity for special type of markers to extrapolate distance and orientation to the landing platform. Although this approach has been proven successful, allowing for faster experimentation and simplification of the training pipeline, it is less generalizable. In this paper we want to explore how we can learn reinforcement learning control policies in simulation utilizing computer vision to localize our target, without needing to modify the pipeline from simulation to real-world deployment, therefore reducing the simulation-to-reality gap.

3 Methodology

In this study we aim to train a UAV to autonomously land on a restricted surface using a joint Computer Vision and Reinforcement Learning policy. We are therefore interested in simulators that allow for an easy interface with the OpenAI Gym API to formulate our task in a framework compatible with RL algorithms and wrappers.

Table 1: PyBullet and other Simulators

Simulators	Physics Engine	Rendering Engine	Language	Synchro/Steppable Physics & Rendering	RGB, Depth and Segmentation Views	Multiple Vehicles	Gym API	Multi-agent Gym-like API
PyBullet-drones	PyBullet	OpenGL3	Python	✓	✓	✓	✓	✓
AirSim	Ad hoc	Unity	C++	✓	✓	✓	W/o Vision	✗
Flightmare	FastPhysicsEngine	UE4	C++	✗	✓	✓	✗	✗
CrazyS	Gazebo	OGRE	C++	✓	No Segmentation	✗	✗	✗

The table 1 found in [18] demonstrates important features that differentiate available simulation software. Out of all the simulators only PyBullet-drones and AirSim could be operated using Python. AirSim, Unity ML-agents, Gazebo and Webots seem to have the largest community support [18] with AirSim offering a well structured documentation as well. Flightmare and ML-agents simulators both based on Unity offer semi-realistic graphics whereas the only simulator that offered photo-realistic graphics was AirSim, based on the Unreal Engine. Since the end-goal of this work is to impose as little assumptions as possible when transferring the policy from simulation to the real world only these three simulators could realistically be considered. When deciding between AirSim and Unity ML-agents or Flightmare, we chose AirSim as the graphics quality of Unreal Engine was superior to Unity, it was very well documented with an API written in Python and had a bigger active community supporting the project.

3.1 Dataset

To generate our training dataset for the object detector we performed several flights with the DJI Mavic-Air 2 model and captured top-down videos from the drone at various locations. Then we extracted frames from the videos, at a fixed rate, to generate a dataset with images of the portable charging station depicted at 1 and 2. We then used the Albumentations library to augment the original 1600 images, generating an additional 3200 images for training. After testing our algorithm on the validation set and ensuring its validity, we further collected 1200 more top-down images of the landing pad from within the simulation at various altitudes and applied similar augmentations to those images using the same library.

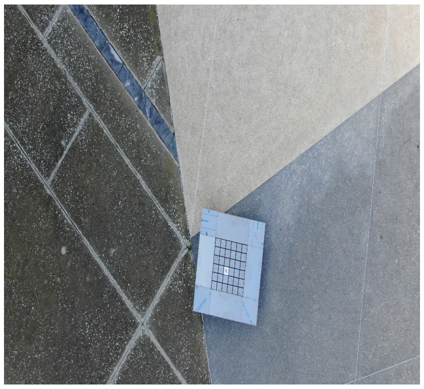


Fig. 1: Landing pad

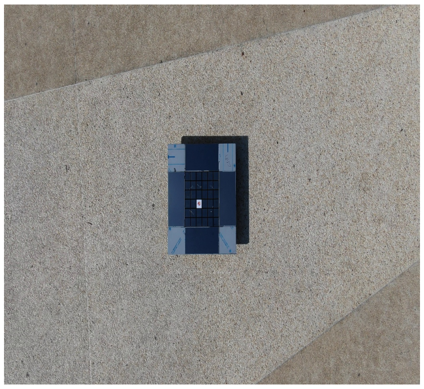
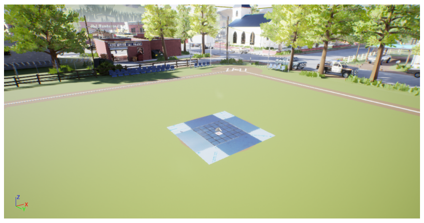
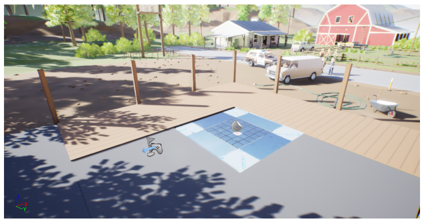


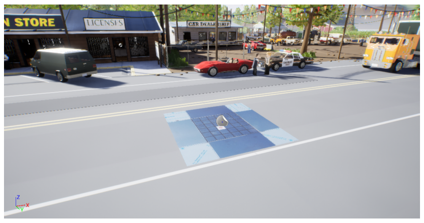
Fig. 2: Landing pad

3.2 Experimental Setup

A city-like environment was installed in Unreal Engine through Epic Games Marketplace and imported as content in the Blocks environment. We then created a custom object in the environment using the tile object template and changed its texture to be that of a top-down landing pad image based on a high-definition image we imported from our dataset. In an increasing background complexity manner, we have placed landing pads at a baseball field Figure (3a), a construction site Figure (3b) and a roadway Figure (3c) to further demonstrate the generalizability of our object detector.


(a) Baseball field background


(b) Construction site background



(c) Roadway background

Fig. 3: Landing pad at various backgrounds

To localize the landing pad we have chosen the Yolov8 algorithm as it is the latest iteration of the YOLO architecture and achieves state-of-the-art results in real time inference, as well as utilizing an anchor-free design [11]. After training on a mixture of real and simulated data (fig. 4) for 30 epochs we achieve a map@0.5 of 98.9%. We later validated the performance of our detector on an out-of-sample video to ensure robustness (fig. 5).

3.3 Algorithms

Having validated the object detector we had to decide on a control policy. We chose a Reinforcement Learning approach as it generalizes better to the unpredictable dynamics of the real world. OpenAI gym environments offer the ability to formulate problems in a Reinforcement Learning framework [4]. We

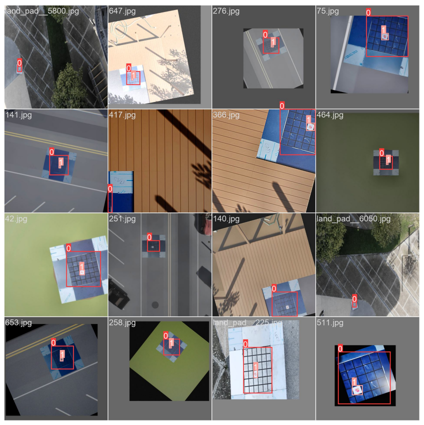


Fig. 4: Training data

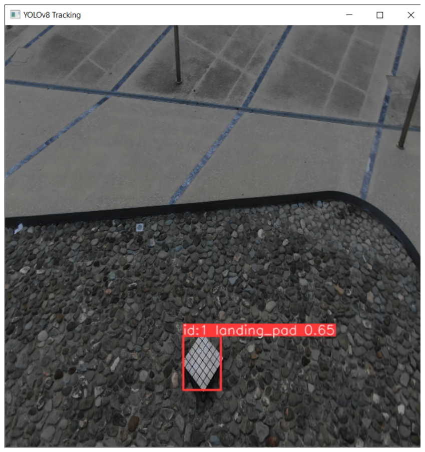


Fig. 5: Validation video sample

subsequently, register a custom Gym environment that allows us to take image observations as inputs and select discrete actions as output.

We make use of out-of-the-box implementation of single-agent algorithms, particularly PPO and LSTM-PPO, offered by [22]. Proximal Policy Optimization (PPO) is an actor-critic policy that compares the probability of an action to be sampled under the current policy over the old policy and increases the probability of the action in the current state if the reward is positive or reduces the probability of the action in the current state if the reward is negative. To prevent the policy from changing too drastically, it utilizes a clipping function, similarly to trust regions, to limit how much the policy parameters are updated. [24]

PPO:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta) * \hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon) * \hat{A}_t)] \tag{1}$$

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \tag{2}$$

A PPO policy cannot correlate consecutive observations unless the observations are provided in a stacked manner or the policy is augmented with temporal-aware modules such as an LSTM module. LSTM-PPO optimizes the same objective function as PPO outlined by (1)

3.4 Reward function

There are two approaches to guide the agent to the desired objective. A sparse reward approach; a reward is given only when the objective is achieved and a dense reinforcement signal, that is given additionally to the terminal reward at each time step, to nudge the agent towards the objective faster.

A dense reward signal provides frequent feedback to the agent and speeds up training, ensuring that the state space that stimulates learning is reached no matter how limited it might be. On the other hand auxiliary rewards given to the agent could lead to unpredictable behaviours that are followed when the agent avoids the terminal state to maximize it's cumulative reward by exploiting those signals.

In our setup if the drone is airborne and the reward shaping is in use, then the drone receives a tiny reward for each timestep that is within landing distance in the xy-plane from the platform, otherwise it receives a small penalty. When the UAV lands if the landing occurred within the precision landing zone it receives a positive reward of at least +1 otherwise a -1 reward.

$$R = \sum_{t=0}^{t=T} r_t \tag{3}$$

Out of bounds or time limit

$$r_{terminal} = -1, |\boldsymbol{r}| > bounds \text{ and } t > 120 \tag{4}$$

Reward Shaping

$$r_t = \begin{cases} 1 \times 10^{-2}\,, & |\boldsymbol{r}| \leq pz \\ -1 \times 10^{-2}\,, & \text{otherwise} \end{cases} \tag{5}$$

Landing reward

$$r_{terminal} = \begin{cases} 1, & |\boldsymbol{r}| \leq pz \\ -\min(0.2 * |\boldsymbol{r}|, 1), & \text{otherwise} \end{cases} \tag{6}$$

where pz is the precision zone radius.

3.5 Pre-processing Steps

The observations received from the environment are pre-processed to be transformed to a distance metric from the image centre and passed to the RL policy as a feature vector. The policy selects an action to be performed by the quadrotor agent and an associated reward with said action.

Initial experiments with a single frame resulted in policies that failed to generalize when tested on multiple environments. To compensate for this, frame-stacking was implemented and a recurrent version of PPO was used to enhance the temporal nature of the captured data.

At previous stages of experimentation we were utilizing RGB images of 640×640 or 416×416 resolution, similar to our dataset. The image resolution was later reduced to 320×320 as detection performance did not deteriorate and it increased overall simulation speed, leading to reduced training time.

To transform the normalized object coordinates, captured in the video stream, to an objective that could be optimized we followed a series of essential pre-processing steps. We had set a preferred detection flag that prioritized the inner target whenever possible to increase precision from the target centre and subsequently selected the one with the highest confidence score. We then subtracted the coordinates of the bounding box centre from the image centre coordinates and produced a euclidean distance in the x-y plane.

$$\begin{aligned} &\tilde{x} = \frac{x_c}{width} \ , \ \tilde{y} = \frac{y_c}{height} \ , \ \tilde{z} = \frac{z_t}{z_0} \\ &\hat{x} = (\tilde{x} - 0.5) \cdot scaling \\ &\hat{y} = (\tilde{y} - 0.5) \cdot scaling \\ &d = \sqrt{\hat{x}^2 + \hat{y}^2} \end{aligned} \tag{7}$$

Eq. (7) demonstrates the transformation of bounding boxes to a distance metric. x_c represents the x-coordinate of the bounding-box centre which is normalized by dividing by the image width. Similarly, y_c is the y-coordinate of the bounding-box centre divided by the image height. The height of the drone is normalized by dividing the current height of the drone z_t by the initial height z_0; the height where the landing maneuver was initiated. Subsequently, we calculate the distance between the normalized bounding box coordinates and the image centre coordinates by multiplying the remainder with the scaling factor, a value that was empirically found to correspond to the ratio of UE4 distance units and centimeters. Finally, we derive the euclidean distance from the target centre in the xy-plane.

Another issue that had to be addressed was that of partial and missing observations due to detection failures. This resulted in policies not generalizing well enough. To address partial observations, we relabelled the landing pad with bounding boxes only outlining the inner portion of the landing pad. To improve tracking performance, a ByteTrack tracker was additionally attached to the Yolov8 object detector.

In addition, whenever our tracking pipeline failed to produce a detection we used previous observations along with the distance travelled, given it's last action and its duration, to better estimate the new bounding box coordinates, according to the linear motion equations Eq. (8).

$$\begin{aligned} a_{t-1} &= \left[V_x, V_y, V_z\right]^T \\ x &= V_x \cdot t \cdot scaling \\ y &= V_y \cdot t \cdot scaling \end{aligned} \tag{8}$$

a_{t-1} corresponds to the action taken during the last time step, t denotes the duration of our last action and the *scaling* factor is an empirically derived constant that corresponds to the ratio between UE4 distance units and cm. These values are added to the last known distance between bounding box and image centre.

To transform between in-simulation units and real distances we use the real measurements of the landing pad which were 48×48 cm for the inner part and

96 × 96 cm when accounting for the outer frame as well and the in simulation distance of the landing pad. A ratio of 4.48 is obtained when matching cm to in-simulation distance units. Any distance less than 68cm in the xy-plane is considered to be a successful landing. This cutoff point was selected to match the ratio of UE units to real dimensions multiplied by the landing distances in UE units.

4 Experiments and Results

The experimental setup is visualized by figure 6. At each timestep the AirSim simulator issues an API call through the use of its python client to request an RGB Image from the Unreal Engine environment. The image observation is then passed to the Yolov8 detector and the ByteTrack tracker [31] to extract normalized bounding boxes of observations.

We spawn the drone at a random distance between -3 and 3 units of the inner target for both x and y. The limits visually map to just outside the size of the landing pad. Subsequently, we takeoff until we reach a height of 8 units. We monitor each policy's success rate every 100 episodes and allow for the policies to converge before interrupting training and evaluating their performance. LSTM-PPO and PPO algorithm have different average episode lengths but training converge around 20K-25K episodes. We save the weights of the network periodically and then select the one with the best performance during it's last 100 episodes.

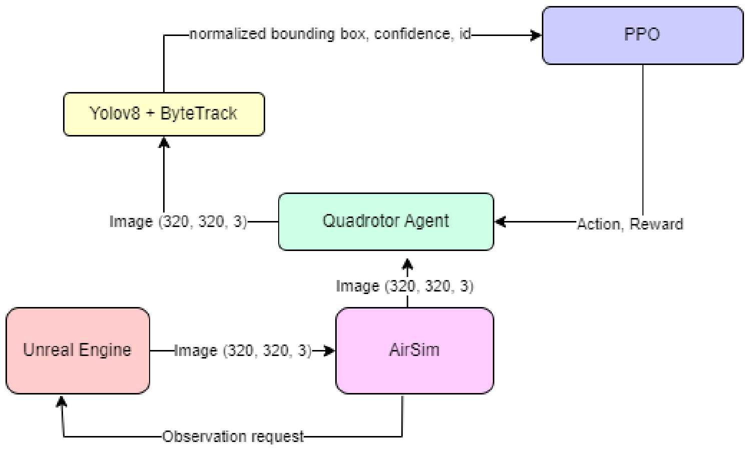


Fig. 6: Experimental Setup

To ensure the reliability of the object detector and minimize the sim-to-real gap, the Yolov8 algorithm was trained on a mixture of real and augmented data generated both in-simulation and real environments and validated in footage

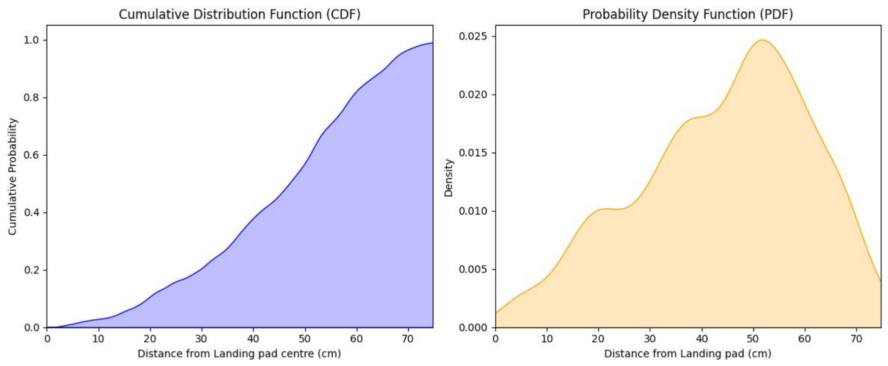


Fig. 7: CDF and PDF of landing distance

taken from real flights. The detector achieves an average map@50 of 0.989% for both inner and outer landing targets.

In our experiments we randomly alternate between the three available environments for a total of 1391 and 817 landings for the PPO and LSTM-PPO algorithms. We experiment with PPO and LSTM-PPO algorithms, as they are amongst the state-of-the-art RL policies. We reach a landing success rate of 94.25% and 94.86% respectively. The mean landing distance is $44+/-17$ cm and $45+/-19$ cm.

Fig. 7 displays the cumulative and probability density functions of the landing distance to the target centre. The distance is measured as the centre of the drone to the centre of the platform. The cumulative density function demonstrates that 94.25% of landings occur at a distance less than 68 cm. The probability distribution function highlights the distance at which the UAV is more probable to land from the platform centre. The highest PDF is around the mean landing distance of $44+/-17$ and $45+/-19$ cm for each algorithm.

Fig. 8 demonstrates the trajectory from a single run that the UAV takes to land at the landing pad. The initial stages of the trajectory are characterized by a seamless flow, gradually transitioning into circular maneuvers as the UAV approaches its target. Implementing an appropriate objective that gradually decreases the velocity as the UAV nears the landing area would enhance the overall smoothness of its trajectory.

5 Discussion

In our experiments before training the RL control policy, we first trained and validated the object detector using real-world top-down images collected from drone flights. Once it demonstrated satisfactory performance in real-world scenarios, we further validated its effectiveness in simulated environments. Our results indicate that pretrained object detectors can be used as components withing a control policy learning framework. This approach allows to remove any assump-

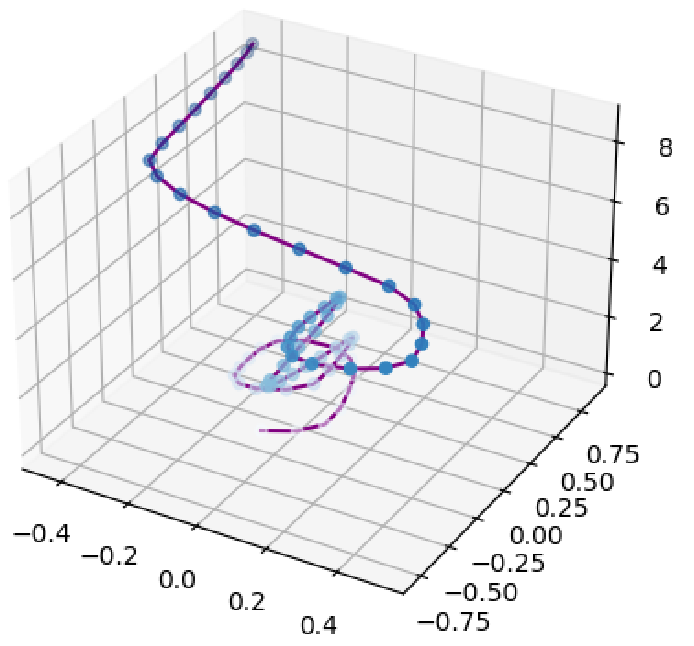


Fig. 8: Landing trajectory

tions on the appearance and the localization of the target when moving from simulation to the real world.

Importantly the learned policy from the fixed object detection model managed to provide robust results and perform autonomous precision landing in three different simulated environments. This is particularly important in order to train policies when targets are not fixed and have different characteristics and are distorted by lighting, occlusions, or viewpoint variations. The learned policies are also more robust to varying altitudes which is often the case in UAV scenarios.

Moreover, the integration of an object detector alongside a robust tracking methodology has been devised to effectively decouple policy learning from feature extraction within a reinforcement learning framework. In doing so, it permits the RL algorithm to operate on normalized bounding boxes of the target, with a vector of observations, irrespective of the image's quality, the target's size or detection ID. This showcases the modular capabilities of the proposed pipeline. For instance the detector could be replaced by another model to perform the same task on a different target and then used the same RL policy, or train the RL policy on a new objective using the same detector to perform a different task or substitute object detection with a different vision-based algorithm altogether such as segmentation.

6 Conclusion

In this paper we have demonstrated that it is possible to perform precision landing of a UAV with a joined computer vision and reinforcement learning policy without relying on specific markers. We aimed to minimize assumptions that would differentiate the simulated environment from the real world as much as possible, therefore we used a pre-trained object detector that was validated on footage collected from drone flights at various backgrounds. We further ensured its robustness by training and testing on a mixture of real and simulated images. We formulate the object detection output in a way that can be utilized by a reinforcement learning algorithm enable them to work together. Because of the modular design of the proposed approach we were able to only train the RL control policy whilst keeping the object detector's parameters unchanged. As future work we aim to perform more experiments in the real-world to validate the learned policies and further investigate the sim-to-real potential.

Acknowledgements

This work is supported by the European Union (i. ERC, URANUS, No. 101088124, and ii. Horizon 2020 Teaming, KIOS CoE, No. 739551). This work is also supported from the Government of the Republic of Cyprus through the Cyprus Deputy Ministry of Research, Innovation and Digital Policy.

References

1. Ang, K.H., Chong, G., Li, Y.: Pid control system analysis, design, and technology. IEEE Transactions on Control Systems Technology **13**(4), 559–576 (2005). https://doi.org/10.1109/TCST.2005.847331
2. Babushkin, A.: jmavsim. https://github.com/PX4/jMAVSim (2013)
3. Bonatti, R., Madaan, R., Vineet, V., Scherer, S.A., Kapoor, A.: Learning controls using cross-modal representations: Bridging simulation and reality for drone racing. CoRR **abs/1909.06993** (2019), http://arxiv.org/abs/1909.06993
4. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym (2016)
5. Collins, J., Chand, S., Vanderkop, A., Howard, D.: A review of physics simulators for robotic applications. IEEE Access **9**, 51416–51431 (2021). https://doi.org/10.1109/ACCESS.2021.3068769
6. Djuknic, G.M., Richton, R.E.: Geolocation and assisted gps. Computer **34**(2), 123–125 (2001)
7. Du, L., Zhang, R., Wang, X.: Overview of two-stage object detection algorithms. In: Journal of Physics: Conference Series. vol. 1544, p. 012033. IOP Publishing (2020)

8. Furrer, F., Burri, M., Achtelik, M., Siegwart, R.: Robot Operating System (ROS): The Complete Reference (Volume 1), chap. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-26054-9_23
9. Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., Marín-Jiménez, M.: Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recognition **47**(6), 2280–2292 (2014). https://doi.org/https://doi.org/10.1016/j.patcog.2014.01.005, https://www.sciencedirect.com/science/article/pii/S0031320314000235
10. Hwangbo, J., Sa, I., Siegwart, R., Hutter, M.: Control of a quadrotor with reinforcement learning. IEEE Robotics and Automation Letters **2**(4), 2096–2103 (2017). https://doi.org/10.1109/LRA.2017.2720851
11. Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., NanoCode012, Kwon, Y., Michael, K., TaoXie, Fang, J., imyhxy, Lorna, Yifu, Z., Wong, C., V, A., Montes, D., Wang, Z., Fati, C., Nadar, J., Laughing, UnglvKitDe, Sonck, V., tkianai, yxNONG, Skalski, P., Hogan, A., Nair, D., Strobel, M., Jain, M.: ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation (Nov 2022). https://doi.org/10.5281/zenodo.7347926, https://doi.org/10.5281/zenodo.7347926
12. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. The International Journal of Robotics Research **32**(11), 1238–1274 (2013)
13. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566). vol. 3, pp. 2149–2154 vol.3 (2004). https://doi.org/10.1109/IROS.2004.1389727
14. Kooi, J.E., Babuska, R.: Inclined quadrotor landing using deep reinforcement learning. CoRR **abs/2103.09043** (2021), https://arxiv.org/abs/2103.09043
15. Liu, P., Chen, A.Y., Huang, Y.N., Han, J.Y., Lai, J.S., Kang, S.C., Wu, T.H., Wen, M.C., Tsai, M.H., et al.: A review of rotorcraft unmanned aerial vehicle (uav) developments and applications in civil engineering. Smart Struct. Syst **13**(6), 1065–1094 (2014)
16. Michel, O.: Webots: Professional mobile robot simulation. Journal of Advanced Robotics Systems **1**(1), 39–42 (2004), http://www.ars-journal.com/International-Journal-of- Advanced-Robotic-Systems/Volume-1/39-42.pdf
17. Morar, A., Moldoveanu, A., Mocanu, I., Moldoveanu, F., Radoi, I.E., Asavei, V., Gradinaru, A., Butean, A.: A comprehensive survey of indoor localization methods based on computer vision. Sensors **20**(9) (2020). https://doi.org/10.3390/s20092641, https://www.mdpi.com/1424-8220/20/9/2641
18. Panerati, J., Zheng, H., Zhou, S., Xu, J., Prorok, A., Schoellig, A.P.: Learning to fly - a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control. CoRR **abs/2103.02142** (2021), https://arxiv.org/abs/2103.02142
19. Pham, H.X., La, H.M., Feil-Seifer, D., Van Nguyen, L.: Reinforcement learning for autonomous uav navigation using function approximation. In: 2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR). pp. 1–6 (2018). https://doi.org/10.1109/SSRR.2018.8468611
20. Polvara, R., Patacchiola, M., Hanheide, M., Neumann, G.: Sim-to-real quadrotor landing via sequential deep q-networks and domain randomization. Robotics **9**(1) (2020). https://doi.org/10.3390/robotics9010008, https://www.mdpi.com/2218-6581/9/1/8

21. Polvara, R., Patacchiola, M., Sharma, S.K., Wan, J., Manning, A., Sutton, R., Cangelosi, A.: Autonomous quadrotor landing using deep reinforcement learning. CoRR **abs/1709.03339** (2017), http://arxiv.org/abs/1709.03339
22. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research **22**(268), 1–8 (2021), http://jmlr.org/papers/v22/20-1364.html
23. Rodriguez-Ramos, A., Sampedro, C., Bavle, H., Moreno, I.G., Campoy, P.: A deep reinforcement learning technique for vision-based autonomous multi-rotor landing on a moving platform. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 1010–1017 (2018). https://doi.org/10.1109/IROS.2018.8594472
24. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR **abs/1707.06347** (2017), http://arxiv.org/abs/1707.06347
25. Shah, S., Dey, D., Lovett, C., Kapoor, A.: Airsim: High-fidelity visual and physical simulation for autonomous vehicles. CoRR **abs/1705.05065** (2017), http://arxiv.org/abs/1705.05065
26. Silano, G., Aucone, E., Iannelli, L.: CrazyS: A Software-In-The-Loop Platform for the Crazyflie 2.0 Nano-Quadcopter. In: 2018 26th Mediterranean Conference on Control and Automation (MED). pp. 352–357 (Jun 2018). https://doi.org/10.1109/MED.2018.8442759
27. Song, Y., Naji, S., Kaufmann, E., Loquercio, A., Scaramuzza, D.: Flightmare: A flexible quadrotor simulator. In: Proceedings of the 2020 Conference on Robot Learning. pp. 1147–1157 (2021)
28. Terven, J., Cordova-Esparza, D.: A comprehensive review of yolo: From yolov1 to yolov8 and beyond. arXiv preprint arXiv:2304.00501 (2023)
29. Tiwari, S.: An introduction to qr code technology. In: 2016 International Conference on Information Technology (ICIT). pp. 39–44 (2016). https://doi.org/10.1109/ICIT.2016.021
30. Zhang, J., Campbell, J.F., Sweeney II, D.C., Hupman, A.C.: Energy consumption models for delivery drones: A comparison and assessment. Transportation Research Part D: Transport and Environment **90**, 102668 (2021). https://doi.org/https://doi.org/10.1016/j.trd.2020.102668, https://www.sciencedirect.com/science/article/pii/S1361920920308531
31. Zhang, Y., Sun, P., Jiang, Y., Yu, D., Weng, F., Yuan, Z., Luo, P., Liu, W., Wang, X.: Bytetrack: Multi-object tracking by associating every detection box. In: European Conference on Computer Vision. pp. 1–21. Springer (2022)