Toward End-to-End Control for UAV Autonomous Landing via Deep Reinforcement Learning

Riccardo Polvara[1]*, Massimiliano Patacchiola[2]*
Sanjay Sharma[1], Jian Wan[1], Andrew Manning[1], Robert Sutton[1] and Angelo Cangelosi[2]

***Abstract*— The autonomous landing of an unmanned aerial vehicle (UAV) is still an open problem. Previous work focused on the use of hand-crafted geometric features and sensor-data fusion for identifying a fiducial marker and guide the UAV toward it. In this article we propose a method based on deep reinforcement learning that only requires low-resolution images coming from a down looking camera in order to drive the vehicle. The proposed approach is based on a hierarchy of Deep Q-Networks (DQNs) that are used as high-end control policy for the navigation in different phases. We implemented various technical solutions, such as the combination of vanilla and double DQNs trained using a form of prioritized buffer replay that separates experiences in multiple containers. The optimal control policy is learned without any human supervision, providing the agent with a sparse reward feedback indicating the success or failure of the landing. The results show that the quadrotor can autonomously land on a large variety of simulated environments and with relevant noise, proving that the underline DQNs are able to generalise effectively on unseen scenarios. Furthermore, it was proved that in some conditions the network outperformed human pilots.**

I. INTRODUCTION

Unmanned Aerial Vehicles (UAVs) are already deployed in various situations such as surveillance [1], agriculture [2], [3], mapping [4], inspection [5] and search and rescue [6]. Recently, they have been taken into account for the transportation and delivery of packages and goods. In this case, one of the most delicate phase is the identification of a fiducial marker and the descending maneuver to ground level. Landing must be done in a limited amount of time and space, using high precision sensing techniques for an accurate control and path planning. Until now this task was performed using hand-crafted features analysis and external sensors (e.g. ground cameras, range scanners, differential GPS, etc.). In this paper we propose instead a different approach, inspired by a recent breakthrough achieved with Deep Reinforcement Learning (DRL) [7]. Our method is based on a hierarchy of Deep Q-Networks (DQNs) taking in input a sequence of low-resolution images acquired by a down-looking camera mounted on the UAV. The DQNs directly communicate with the closed loop flight controller acting as an high level navigator that identifies the position of the marker and moves the drone toward it. The most remarkable advantage of DRL is the total absence of human supervision during the learning process. The quadrotor can autonomously learn what is the best action to perform in order to land.

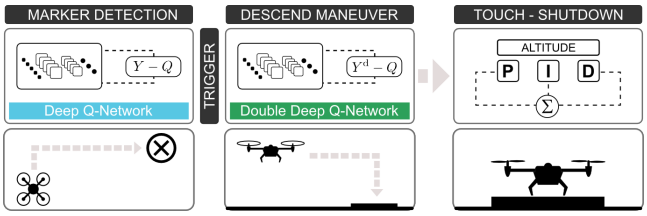


Fig. 1: Components of the proposed landing system. A first DQN takes care of the marker detection on the xy-plane. The second DQN handles the descending maneuver from 20 to 1.5 meters. The last module is a closed loop controller that takes control from 1.5 meter to ground level.

The applicability of DRL in robotics is not straightforward. So far, the research focused on deterministic environment, such as the Atari game suite [7], and the Doom platform [8]. The success of DRL in complex robotics tasks has been limited. In order to tackle the landing problem we introduced different technical solutions. First, we adopted a divide-and-conquer approach splitting the problem in two sub-tasks: landmark detection and vertical descent. Both tasks are addressed by two independent DQNs that are able to call each other through an internal trigger. We encountered different problems, such as the overestimation of the utilities and the reward sparsity. In order to face the overestimation we adopted the *double DQN* architecture proposed in [9]. To solve the reward sparsity we developed a new form of prioritized experience replay called *partitioned buffer replay*. Using a partitioned buffer replay it is possible to split experiences based on their relevance and guarantee the presence of rare transitions in the training batch. An overview of the system is shown in Figure 1.

The overall contribution of this article can be summarized in three points. (i) As far as we know, this work is the first to use an unsupervised learning approach in autonomously landing. The training phase has been completed using low-resolution images, without any direct human supervision or hand-crafted features. For this reason, this method represents a significant improvement compared to previous research. (ii) We introduce new technical solutions such as a hierarchy of deep Q-networks able to autonomously trigger each other, and a new form of prioritized buffer replay. (iii) The proposed method has been used to train a commercial quadrotor

*Both first and second author contributed equally and should be considered co-first authors.

[1]Autonomous Marine System Research Group, School of Engineering, Plymouth University, United Kingdom.
[2]Centre for Robotics and Neural Systems, School of Computing, Electronics and Mathematics, Plymouth University, United Kingdom.

in a variety of simulated environments. The testing phase showed a relevant generalisation on unseen grounds and an overall performance comparable with human pilots.

II. Related work

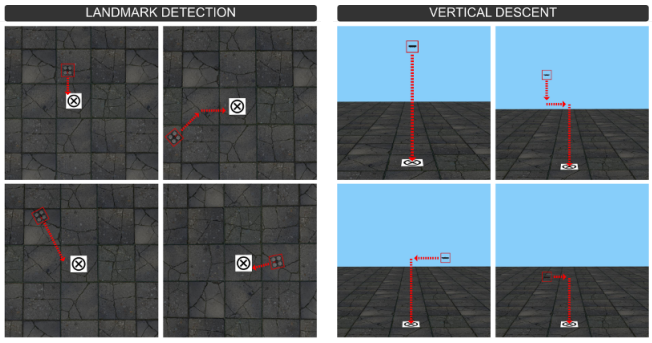
In the following section we offer an overview of the research that has been done in autonomous landing of UAVs. This section helps to compare our method with previous work. A complete description of all the methods available is out of scope so we refer the reader to recent surveys [10], [11]. The landing problem has been investigated from different points of view and using various techniques. For simplicity we group all the different methods in three classes: sensor-fusion systems, device-assisted systems, vision-based systems.

Combining data from multiple sensors is a common expedient used to improve the performances. This is the main strategy used in sensor-fusion systems. An example is offered in [12]. In this article the cameras, GPS and differential GPS data are gathered together for estimating the UAV's relative pose with respect to a landing pad. Similarly, [13] combined data from a monocular camera, odometry and inertial unit to estimate the position of the vehicle in order to land on a moving vessel. In [14] camera and IMU are integrated to build a three dimensional model of the terrain that allowed identifying a safe landing area. A ground-based multisensor fusion system has been proposed in [15]. The system included a pan-tilt unit, an infrared camera and an ultra-wideband radar used to center the UAV in a recovery area.

Device-assisted methods use sensors located on the ground to precisely estimate the pose of the UAV. In [16] the authors proposed a vision system based on two parallel cameras in order to allow the vehicle to land on a runway. Infra-red parallel lamps were adopted in [17]. The camera on the vehicle was equipped with optical filters sensible to infra-red lights and the incoming video stream was processed by a control loop for pose estimation purposes. Finally, a Chane-Vase based approach has been proposed in [18] for ground stereo-vision detection.

The vision-based approaches comprise those techniques that rely on the analysis of vision features to identify and extract the landing pad. An example is given in [19]. The authors used a series of concentric circles to make the marker visible at different distances and resolutions. A similar idea, adopted in [20], relies on parallel computation performed by an on-board GPU for obtaining real-time pose estimation. In [21] a seven-stage vision algorithm was able to reconstruct the international H-shape landing pattern when partially occluded. Finally, [22] and [23] offer a solution to autonomously land on a moving ground platform.

Previous work presents some limitations of which we are going to describe. Sensor-fusion methods rely on expensive sensors that cannot be easily integrated on low-cost or commercial drones. Moreover, most of these methods use GPS signal which is often not available in cluttered real-world scenarios. The device-assisted approaches offer a good pose estimation of the UAV during all the landing manoeuvre thanks to the use of external sensors. On the other hand, this represents a limitation because it is not possible to land in an unknown environment where such devices are not available. Vision-based methods only rely on sensors mounted on-board, more specifically on cameras. Unfortunately, it is often difficult to identify low-level features, especially when the pad is distant, partially obscured or blurred.



(a) (b)

Fig. 2: Representation of landmark detection (a) and vertical descent (b). In (a) the drone has to align to the ground marker. In (b) the drones has to reduce the distance from the marker vertical movements.

The proposed approach aims to address most of the aforementioned problems in a novel way. Our method only relies on a single on-board camera, and it does not need additional sensors or external devices. The use of DQN improves drastically the marker detection and is more robust to projective transformations and occlusions. While we make no claim that our method is better than the existing ones, we believe that our policy is at least competitive with the best models in the literature and highlight the potential of the DRL in UAV navigation.

III. Proposed method

In this section we describe the landing problem in reinforcement learning terms and we present the technical solutions used in our method.

A. Problem definition and notation

As discussed in Section II, there is limited work which attempted to tackle the landing problem using reinforcement learning and in particular DRL. The use of UAVs introduces many complications. Drones move in a three-dimensional space which is expensive to explore. This additional spatial dimension introduces partial observability and occlusions, moreover it causes complications due to perspective variance. The agent can only view a small portion of the environment and it has to deal with projective transformations. For all these reasons, applying DQN to the landing problem is challenging. In previous research, DQN obtained high performances in two-dimensional games where all the problems mentioned above were attenuated or absent.

Here we consider the landing problem as divided in two problems: landmark detection and vertical descent. The landmark detection requires the UAV to explore the xy-plane through horizontal shifts, in order to align itself with the marker. In the vertical descent phase the quadrotor must decrease its altitude through vertical and lateral movements to keep the marker centred within the camera's field of view. A graphical representation of the two phases is reported in Figure 2.

From a formal point of view, we can briefly define reinforcement learning as the problem faced by an agent that learns through trial-and-error interactions with a dynamic environment [24]. In our specific case, the landing problem is modelled as Markov Decision Processes (MDPs). At each time step t the agent receives the state s_t, performs an action a_t sampled from the action space A, and receives a reward r_t given by a reward function $R(s_t, a_t)$. The action brings the agent to a new state s_{t+1} in accordance with the environmental transition model $T(s_{t+1}|s_t, a_t)$. In the particular case faced here the transition model is not given (model free). The goal of the agent is to maximize the discounted cumulative reward called return $R = \sum_{k=0}^{\infty} \gamma^k r_{t+1}$, where γ is the discount factor. Given the current state the agent can select an action from the internal policy $\pi = P(a|s)$. In off-policy learning the prediction of the cumulative reward can be obtained through an action-value function $Q^{\pi}(s, a)$ adjusted during the learning phase in order to approximate $Q^*(s, a)$, the optimal action-value function. In simple cases Q can be represented in tabular form. The tabular approach is generally inadequate to model a large state space due to combinatorial explosion. To solve the problem, function approximation (e.g. artificial neural networks) can be used to represent the action-value function. In this work we use two Convolutional Neural Networks (CNNs) for function approximation following the approach presented in [7]. Recently, CNNs achieved outstanding results in a large variety of problems (see [25] for a review). In the work here presented, the CNNs receive in input a stack of four 84×84 grey-scale images acquired by a down looking camera. The images are then processed by three convolutional layers for feature extraction and two fully connected layers. An example of the input images and the output of the first layer of kernel is provided in Figure 3. As activation function we used the rectified linear unit [26]. The first convolution has 32 kernels of 8×8 with stride of 2, the second layer has 64 kernels of 4×4 with strides of 2, the third layer convolves 64 kernels of 3×3 with stride 1. The fourth layer is a fully connected layer of 512 units followed by the output layer which has a unit for each valid action (backward, right, forward, left, stop, descend, land). Depending on the simulation, we used a sub-set of the total actions available, we refer the reader to Section IV for additional details.

In order to identify the problems that can affect and compromise the learning, it is important to carefully analyze the two phases characterizing the overall landing manoeuvre. The landmark detection phase is performed at a fixed altitude. The UAV aligns its frame with the marker through planar shifts. This expedient significantly reduces the complexity of the task and does not have any impact at the operational level. To adjust θ, the parameters of the DQN, in this phase we used the following loss function:

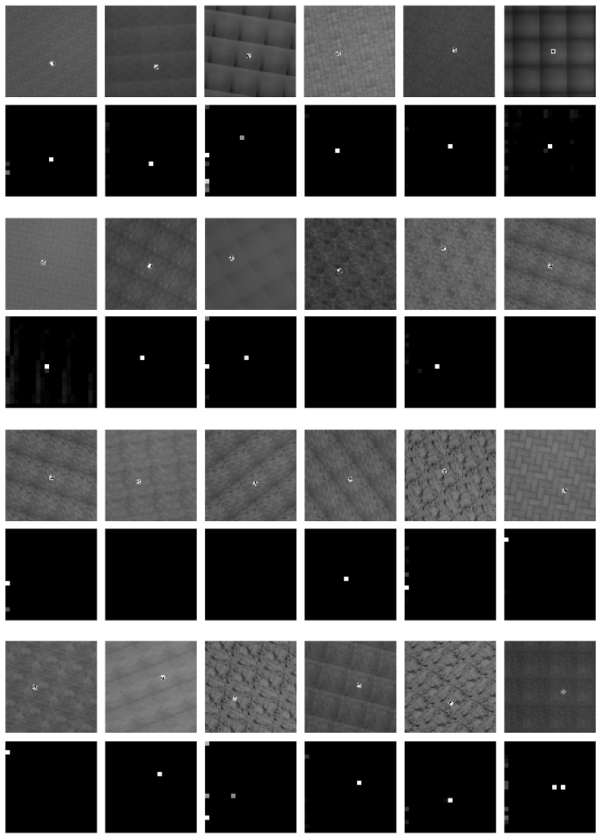


Fig. 3: The greyscale images given as input to the DQN and the feature map generated by the kernels in the first convolutional layer.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)} \left[\left(Y_i - Q(s, a; \theta_i) \right)^2 \right] \quad (1)$$

with $D = (e_1, ..., e_t)$ being a dataset of experiences e $e_t = (s_t, a_t, r_t, s_{t+1})$ used to uniformly sample a batch at each iteration i. The network $Q(s, a; \theta_i)$ is used to estimate actions at runtime, whereas Y_i is the target which is defined as follows:

$$Y_i = r + \gamma \max_{a'} Q(s', a'; \theta_i^-) \quad (2)$$

the network $Q(s', a'; \theta_i^-)$ is used to generate the target and is constantly updated. The use of the target network is a trick that improves the stability of the method. The parameters θ are updated every C steps and synchronized with θ^-. In the standard approach the experiences in the dataset D are collected in a preliminary phase using a random policy. The dataset D is also called buffer replay and it is a way to

randomize the samples breaking the correlation and reducing the variance [27].

The vertical descend phase is a form of Blind Cliffwalk [28] where the agent has to take the right action in order to progress through a sequence of N states. At the end of the walk the agent can obtain a positive or a negative reward. The intrinsic structure of the problem makes extremely difficult to obtain a positive reward because the target-zone is only a small portion of the state space. The consequence is that the buffer replay does not contain enough positive experiences, making the policy unstable. To solve this issue we use a form of buffer replay called partitioned buffer replay, that discriminates between rewards and guarantees a fair sampling between positive, negative and neutral experiences. We are going to describe the partitioned buffer replay in Section III-B. Another issue connected with the reward sparsity is a well known problem called overestimation [29]. During a preliminary research we observed that this phenomenon arose in the vertical descent phase. A solution to overestimation has been recently proposed and has been called double DQN [9]. The target estimated through double DQN is defined as follows:

$$Y_i^{\mathrm{d}} = r + \gamma \; Q(s', \underset{a'}{\operatorname{argmax}} \; Q(s', a'; \theta_i); \theta_i^-) \tag{3}$$

Using this target instead of the one in Equation 2 the divergence of the DQN action distribution is mitigated resulting in faster convergence and increased stability.

B. Partitioned buffer replay

Markov Decision Processes characterized by a sparse and delayed reward make difficult for an agent to obtain a positive feedback. This situation leads to unbalanced experiences within the buffer replay. For instance, neutral experiences are sampled more frequently compared to positive and negative ones. To solve this issue has been proposed to divide the experiences in two buckets with different priority [30]. Our approach is an extension of this method to K buckets. Another form of prioritized buffer replay has been proposed in [28], in which the authors suggested to sample important experiences more frequently. The prioritized replay estimates a weight for each experience based on the temporal difference error. Experiences are sampled with a probability proportional to the weight. The limitation of this form of prioritization is that it introduces additional overhead that may not be justified for applications were there is a clear distinction between positive and negative rewards. Moreover, this method requires $O(\log N)$ to update the weights in the priority queue.

In Section III-A we defined $D = (e_1, ..., e_t)$ being a dataset of experiences $e = (s, a, r, s')$ used to uniformly sample a batch at each iteration i. To create a partitioned buffer replay we have to divide the reward space in K partitions:

$$R = R(s, a) \rightarrow \text{Im } R \; = R_1 \cup ... \cup R_K \tag{4}$$

For any experience e_i we associate its reward $r_i = r(e_i)$ and we define the Kth buffer replay:

$$D_K = \{(e_1, ..., e_N) : r_1, .., r_N \in R_K\} \tag{5}$$

The batch used for training the policy is assembled picking experiences from each one of the K datasets with a certain fraction $\rho \in \{\rho_1, ..., \rho_K\}$.

In our particular case we have $K = 3$, meaning that we have three datasets with D^+ containing experiences having positive rewards, D^- containing experiences having negative rewards, and $D^\sim$ for experiences having neutral rewards. The fraction of experiences associated to each one of the dataset is defined as ρ^+, ρ^-, and $\rho^\sim$.

C. Hierarchy of DQNs

The method we propose uses a hierarchy of DQNs for addressing the different phases involved in the landing. Similarly to a finite-state machine, the global policy is divided into sub-modules each one governed by a standalone DQN. The networks are able to automatically call each other in specific portions of the state space. The advantages of this method are twofold. On one hand we reduce the complexity of the task with a divide-and-conquer approach. On the other hand, the use of function approximation is confined in specific sandboxes making their use in robotic applications safer.

The overall landing problem can be described by a three-stages process: landmark detection, descend manoeuvre, and touchdown. We already described in Section III-A the first two phases. The touchdown corresponds to gradually reduce the motors power in the last few centimeters of the landing. This article is focused only on the first two stages, representing the most complex parts of the landing procedure. A graphical representation of a hierarchical state machine is represented in Figure 1. The first DQN, responsible for performing marker detection, is trained to receive a positive reward when the trigger action was enabled within a target area. A negative reward was instead given if the trigger was activated outside the target area. The second network, responsible for the vertical descent, is trained following the same concept. Only once the two networks have been trained is possible to assemble the state-machine pipeline.

IV. Experiments

The following section aims to introduce the methodology and the results obtained after training and testing the two DQNs. Section IV-A is reserved to the methodology and the results obtained in the landmark detection phase. In Section IV-B is presented the second series of simulation concerning the vertical descent phase. Both training and testing are performed within the same simulator (Gazebo 7 and ROS Kinetic) using the same vehicle (Parrot AR Drone 2). The simulator is a fork of the one used in [31] and it is freely available on our repository[1]. Here it is necessary to point out that the physics of the system made

[1]https://github.com/pulver22/QLAB

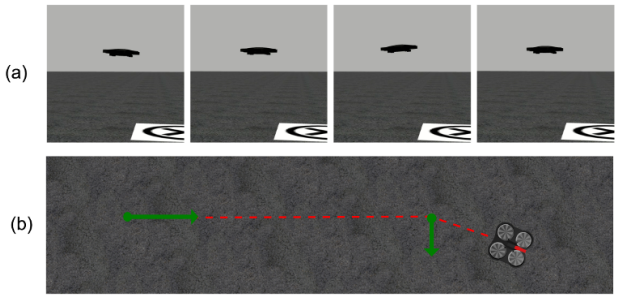


Fig. 4: Illustration of the vehicle inertia. In (a) is represented the oscillatory inertia for roll and pitch, which is visible when the drone starts and stops. In (b) is represented the direction of the drone moving in a straight line (dashed red) and the diagonal trajectory acquired due to the addiction of two forces (green arrows).

extremely challenging to apply DRL to the landing problem. An oscillatory effect during accelerations and decelerations made the drone swinging on the roll and pitch axes (Figure 4a). This effect generated artifacts in the image acquisition of the down-looking camera. Moreover a summation of forces effect introduced a substantial shift in the trajectory (Figure 4b). The shift increased the complexity of the action space introducing an high variance. The DRL algorithm has to deal with this source of noise.

A. *First series of simulations*

In the first series of simulations we trained and tested the DQNs for the marker detection phase. We considered two networks having the same structure and we trained them in two different conditions. The first network was trained with a uniform asphalt texture (DQN-single), whereas the second network was trained with multiple textures (DQN-multi). The ability to generalise to new unseen situations is very important and it should be seriously taken into account in the landing problem. Training the first network on a single texture is a way to quantify the effect of a limited dataset on the performance of the agent. In the DQN-multi condition the networks were trained using seven different groups of textures: asphalt, brick, grass, pavement, sand, snow, soil (Figure 5). These networks should outperform the ones trained in the condition with single texture.

To simplify the UAV movements we only allowed translation on the xy-plane. At each episode the drone started at a fixed altitude of 20 m that was maintained for the entire flight. This expedient was useful for two reasons: it significantly reduced the state space to explore, and it allowed visualizing the marker in most of the cases giving a reference point for the navigation. In a practical scenario this solution does not have any impact on the flight, the drone is kept at a stable altitude and the frames are acquired regularly. To stabilize the flight we introduced discrete movements, meaning that each action was repeated for 2 seconds and then stopped leading to an approximate shift of 1 meter similarly to the no-operation parameter used in [7]. The frames from

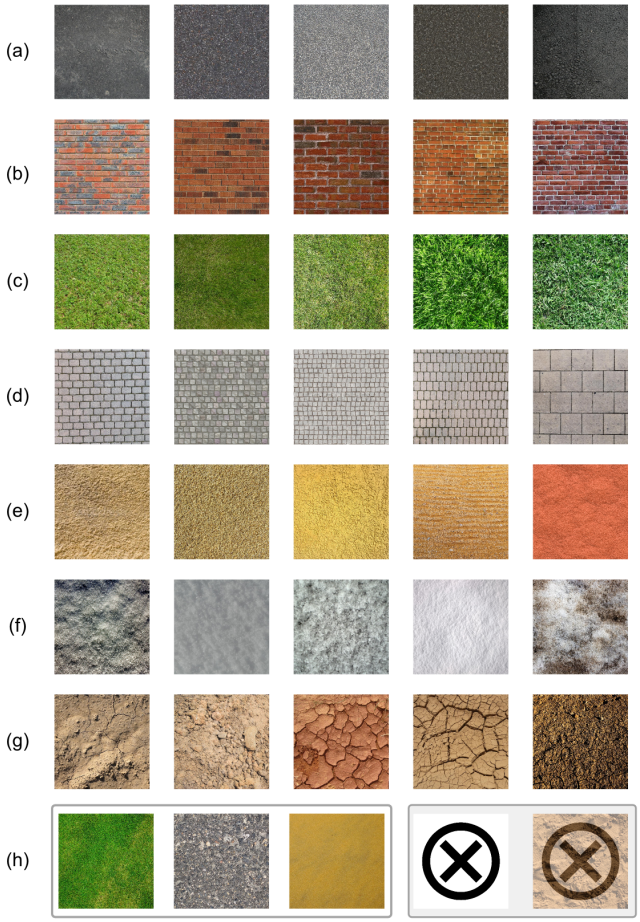
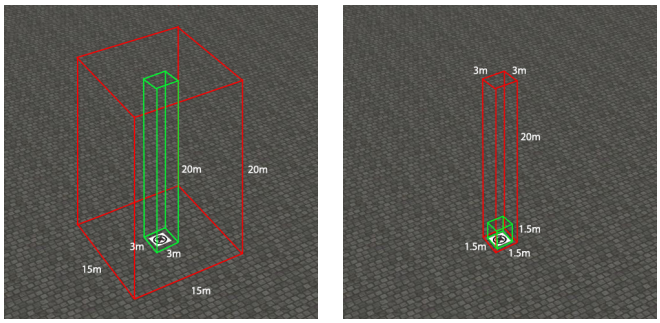


Fig. 5: Textures used during the training phase. Only a subsample of the 91 textures is showed here. From top to bottom there is: asphalt (a), brick (b), grass (c), pavement (d), sand (e), snow (f), soil (g). In (h) is showed a sample of the test set (left), the marker and corrupted marker (right).

the camera were acquired between two actions, when the drone had a constant speed. This expedient helped stabilizing convergence reducing perspective errors.

1) Methods: The training environment was represented by a uniform texture of size 100×100 m with the landmark positioned in the center. The environment contained two bounding boxes. At the beginning of each episode the drone was generated at 20 m of altitude inside the perimeter of the larger bounding box ($15 \times 15 \times 20$ m) with a random position and orientation. A positive reward of 1.0 was given when the drone activated the trigger in the target-zone, and a negative reward of -1.0 was given if the drone activated the trigger outside the target-zone. A negative cost of living of -0.01 was applied in all the other conditions. A time limit of 40 seconds (20 steps) was used to stop the episode and start a new one. In the DQN-multi condition the ground texture was changed every 50 episodes and randomly sampled between the 71 available. The target and policy networks were synchronized every 10000 frames. The agent had five possible actions available: forward, backward, left, right, land-trigger. The



(a) (b)

Fig. 6: Bounding boxes for landmark detection (a) and vertical descent (b). The drone is generated in the red box at the beginning of each episode. The green box (target-zone) gives the positive reward.

action was repeated for 2 seconds then the drone was stopped and a new action was sampled. The buffer replay was filled before the training with 4×10^5 frames using a random policy. We trained the two DQNs for 6.5×10^5 frames. We used an ϵ-greedy policy with ϵ decayed linearly from 1.0 to 0.1 over the first 5×10^5 frames and fixed at 0.1 thereafter. The discount factor γ was set to 0.99. As optimizer we used the RMSProp algorithm [32] with a batch size of 32. The weights were initialized using the Xavier initialization method [33]. The DQN algorithm was implemented in Python using the Tensorflow library [34]. Simulations were performed on a workstation with an Intel i7 (8 core) processor, 32 GB of RAM, and the NVIDIA Quadro K2200 as graphical processing unit. On this hardware the training took 5.2 days to complete.

To test the performance of the policies we measured the landing success rate of both DQN-single and DQN-multi in a new environment using 21 unknown textures. We also measured the performances of a random agent, human pilots and the AR-tracker proposed in [35] on the same benchmark. The random agent data has been collected sampling the actions from a uniform distribution at each time step. The human data has been collected using 7 volunteers. The subjects used a space-navigator mouse that gave the possibility to intuitively and naturally control the drone in the three dimensions. In the landmark detection test the subjects had to align the drone with the ground marker and trigger the landing procedure when inside the target-zone. A preliminary training allowed the subject to familiarize with the task. After the familiarization phase the real test started. The subjects performed five landing attempts for each one of the 21 textures contained in the test set (randomly sampled). A time limit was applied accordingly to the procedure used for testing our algorithms. A negative reward was given when the subjects triggered the landing outside the target-area.

2) Results: The results for both DQN-single and DQN-multi show that the agents were able to learn an efficient policy for maximizing the reward. In both conditions the reward increased stably without any anomaly (Figure 7). The accumulated reward for the DQN-single condition reaches an higher value in the last iterations. The results of the test phase are summarized in Figure 7 (top). The bar chart compares the performances of the DQN-single, DQN-multi, human subjects, random agent and the AR-tracker. The AR-tracker has the highest reported landing success rate with an overall accuracy of 95%, followed by the DQN-multi with a value of 89%. The score obtained by the agent trained on a single texture (DQN-signle) are significantly lower (38%). The human performance is close to the DQN-multi (86%). The performance on the different classes of textures shows that the DQN-multi obtained top performances in most of the environments. The DQN-single had good performances only on two textures: asphalt and grass. We verified using a two-sample t-test if the difference between DQN-multi and human pilots was statistically significant. The results showed that the difference is significant ($t = 2.37$, $p < .05$) and the DQN outperformed humans. It is possible to further analyze the DQN-multi policy observing the action-values distribution in different states (Figure 8). When the drone is far from the marker the DQN for landmark detection penalizes the landing action. However when the drone is over the marker this utility significantly increases triggering the vertical descent state. In order to better compare the DQN-multi with the AR-tracker, we performed an experiment using the corrupted marker showed in Figure 5-h. We noted a drop in performance from 94% to 0% in the the AR-tracker, due to the failure of the underlying template matching algorithm in detecting the corrupted marker at long distances. Differently, the DQN-multi performed fairly well, with a limited drop in performance from 89% to 81%.

B. Second series of simulations

In the second series of simulations we trained and tested the DQNs specialized in the vertical descend. To encourage a vertical descend during the ϵ-greedy action selection we sampled the action from a non-uniform distribution were the descend action had a probability of ρ and the other N actions a probability $\frac{1-\rho}{N}$. We used *exploring-start* to generate the UAV at different altitudes and to ensure a wider exploration of the state space. Instead of the standard buffer replay we used the partitioned buffer replay described in Section III-B. We trained two networks, the former in a single texture condition (DQN-single) and the latter in multi-texture condition (DQN- multi).

1) Methods: The training environment was represented by a flat floor of size 100×100 m with the landmark positioned in the center. The state-space in the vertical descend phase is significantly larger than in the marker detection and exploration is expensive. For this reason we reduced the number of textures used for the training, randomly sampling 20 textures from the 71. We hypothesize that using the entire training set lead to better performance. The action space available was represented by five actions: forward, backward, left, right, down. A single action was repeated for 2 seconds leading to an approximate shift of 1 meter (constant speed of 0.5 m/s). The descend action was performed at a lower

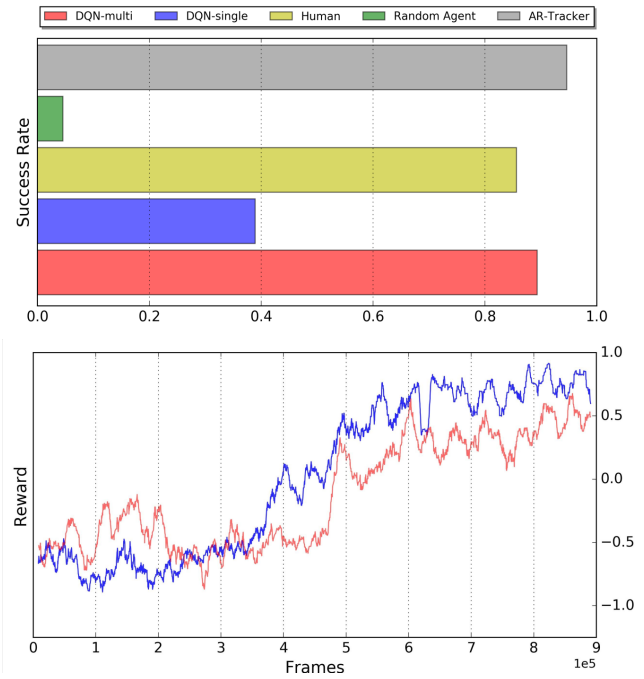


Fig. 7: Results of the first series of simulations. Top: detection success rate. Bottom: accumulated reward per episode for DQN-single (blue line) and DQN-multi (red line).

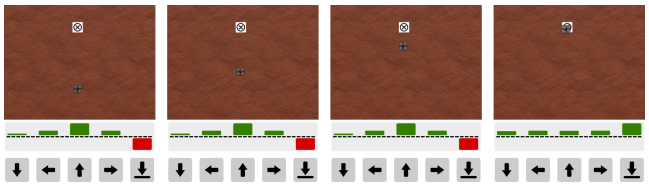


Fig. 8: Snapshots representing the landmark detection. The bottom bar is the utility distribution of the actions. The trigger command has a negative utility (red bar) when the drone is far from the marker.

speed of 0.25 m/s to reduce undesired vertical shifts. For the partitioned buffer replay we chose $\rho^{+} = 0.25$, $\rho^{-} = 0.25$, and $\rho^{\sim} = 0.5$, meaning that 8 positive experiences and 8 negative experiences were always guaranteed in the batch of 32. A time limit of 80 seconds (40 steps) was used to stop the episode and start a new one. The drone was generated with a random orientation inside a bounding box of size $3 \times 3 \times 20$ m at the beginning of the episode. This bounding box corresponds to the target area of the landmark detection phase described in Section IV-A.1. A positive reward of 1.0 was given only when the drone entered in a target-zone of size $1.5 \times 1.5 \times 1.5$ m, centered on the marker. If the drone descended below 1.5 meter outside the target-zone a negative reward of -1.0 was given. A cost of living of -0.01 was applied at each time step. The same hyper-parameters described in Section IV-A.1 were used to train the agent. In addition to the hardware mentioned in Section IV-A.1, we also used a separate machine to collect preliminary experiences. This machine is a multi-core workstation with

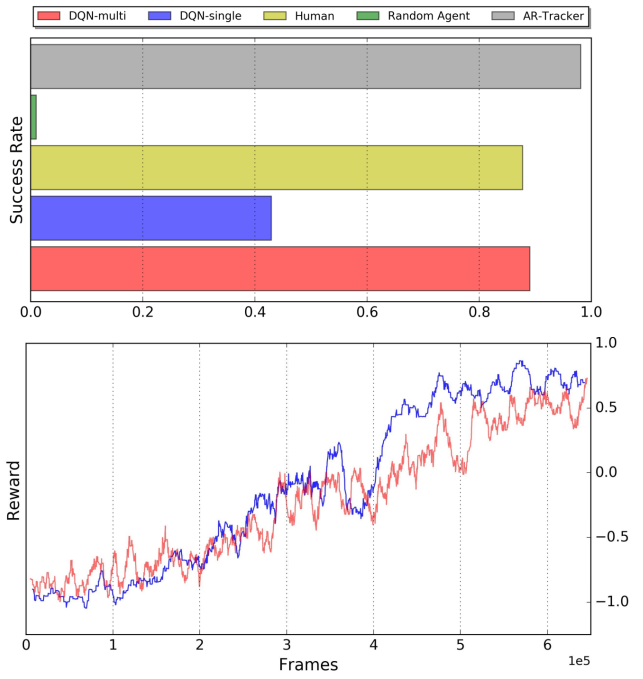


Fig. 9: Results of the second series of simulations. Top: descending success rate. Bottom: accumulated reward per episode for DQN-single (blue line) and DQN-multi (red line).

32 GB of RAM and a GPU NVIDIA Tesla K-40.

Before the training, the buffer replay was filled using a random policy with 10^6 neutral experiences, 5×10^5 negative experiences and 6.2×10^4 positive experiences. We increased the number of positive experiences using horizontal/vertical mirroring and consecutive 90 degrees rotation on all the images stored in the positive partition. This form of data augmentation increased the total number of positive experiences to 5×10^5.

The networks were tested on the same 21 unseen textures used in the marker detection test. Performance of random agent and human pilots has also been collected. The human data has been obtained using a sample of 7 subjects. The subjects had to adjust the vertical position of the drone in order to move toward the marker and obtain a positive reward. The same procedure described in Section IV-A.1 has been used.

2) Results: The results achieved show that both the DQNs were able to learn the task. The accumulated reward per episode showed in Figure 9 (bottom), increased stably in both DQN-single and DQN-multi. The results of the test phase are summarized in Figure 9 (top). The bar chart compares the performances of the DQN-single, DQN-multi, human subjects, random agent and an AR-tracker. Also in this case, the AR-tracker has the highest success rate with an overall accuracy of 98%. The DQN-multi has the same performance obtained in the marker detection phase (89%). The human pilots follow with a score of (87%). The DQN-single score is significantly lower (43%) confirming that the size of the

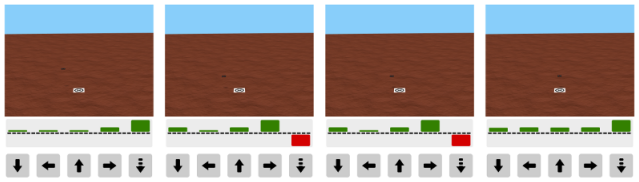


Fig. 10: Snapshots representing the vertical descent phase. The bottom bar is the utility distribution of the actions. The descend command has a negative utility (red bar) when the drone is close to the ground but far from the marker.

training set is an important factor to take into account in order to enhance performances. We performed a t-test to verify if the performances of human pilots and the DQN-multi were significant. The test showed that there is not a significant difference between the two groups ($t = 0.79$, $n.s.$). It is possible to further analyze the DQN-multi policy observing the action-values distribution in different states (Figure 10). When the drone is far from the marker the DQN for landmark detection penalizes the landing action. However when the drone is over the marker this utility significantly increases triggering the landing. Also for the descending phase we compared the performance of the DQN-multi with the AR-tracker using a corrupted marker. This experiment confirmed a drop in performance from 98% to 0% for the AR-tracker due to the failure of the underlying template-matching algorithm. The DQN-multi, on the other hand, proved to be more robust to marker corruption with a limited drop in performance from 89% to 51%.

V. Conclusions and Future Work

In this work we used DRL to realise a system for the autonomous landing of a quadrotor on a static pad. The main modules of the system are two DQNs that can control the UAV in two delicate phases: landmark detection and vertical descent. The two DQNs have been trained in different environments and with relevant noise. We showed that the system can achieve performances comparable with humans and state-of-the-art AR-tracker in the marker detection and in the vertical descent phases. Moreover we showed that we can train robust networks for navigation in large three-dimensional environments by training on multiple maps with random textures. Future work should mainly focus on bridging the reality gap. The reality gap is the obstacles that makes it difficult to implement many robotic solutions in real world. This is especially true for DRL where a large number of episodes is necessary in order to obtain stable policies. Recent research worked on bridging this gap using domain transfer techniques. An example is domain randomization [36], a method for training models on simulated images that transfer to real images by randomizing rendering in the simulator. In conclusion, the results obtained are promising and show that it is possible to apply DQN to complex problems such as the landing one. However further research is necessary in order to reach stable policies which can effectively work in a wide range of conditions.

Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Tesla K40 GPU used for this research.

References

[1] L. Geng, Y. Zhang, J. Wang, J. Y. Fuh, and S. Teo, "Mission planning of autonomous uavs for urban surveillance with evolutionary algorithms," in *Control and Automation (ICCA), 2013 10th IEEE International Conference on*. IEEE, 2013, pp. 828–833.

[2] S. Herwitz, L. Johnson, S. Dunagan, R. Higgins, D. Sullivan, J. Zheng, B. Lobitz, J. Leung, B. Gallmeyer, M. Aoyagi, *et al.*, "Imaging from an unmanned aerial vehicle: agricultural surveillance and decision support," *Computers and electronics in agriculture*, vol. 44, no. 1, pp. 49–61, 2004.

[3] G. Grenzdörffer, A. Engel, and B. Teichert, "The photogrammetric potential of low-cost uavs in forestry and agriculture," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 31, no. B3, pp. 1207–1214, 2008.

[4] M. Nagai, T. Chen, R. Shibasaki, H. Kumagai, and A. Ahmed, "Uav-borne 3-d mapping system by multisensor integration," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 47, no. 3, pp. 701–708, 2009.

[5] N. Metni and T. Hamel, "A uav for bridge inspection: Visual servoing control law with orientation limits," *Automation in construction*, vol. 17, no. 1, pp. 3–10, 2007.

[6] S. Waharte and N. Trigoni, "Supporting search and rescue operations with uavs," in *Emerging Security Technologies (EST), 2010 International Conference on*. IEEE, 2010, pp. 142–147.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[8] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "Vizdoom: A doom-based ai research platform for visual reinforcement learning," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–8.

[9] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning." in *AAAI*, 2016, pp. 2094–2100.

[10] W. Kong, D. Zhou, D. Zhang, and J. Zhang, "Vision-based autonomous landing system for unmanned aerial vehicle: A survey," in *Multisensor Fusion and Information Integration for Intelligent Systems (MFI), 2014 International Conference on*. IEEE, 2014, pp. 1–8.

[11] A. Gautam, P. Sujit, and S. Saripalli, "A survey of autonomous landing techniques for uavs," in *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*. IEEE, 2014, pp. 1210–1218.

[12] S. Saripalli and G. Sukhatme, "Landing on a moving target using an autonomous helicopter," in *Field and service robotics*. Springer, 2006, pp. 277–286.

[13] R. Polvara, S. Sharma, J. Wan, A. Manning, and R. Sutton, "Towards autonomous landing on a moving vessel through fiducial markers," in *2017 European Conference on Mobile Robots (ECMR)*, Sept 2017, pp. 1–6.

[14] C. Forster, M. Faessler, F. Fontana, M. Werlberger, and D. Scaramuzza, "Continuous on-board monocular-vision-based elevation mapping applied to autonomous landing of micro aerial vehicles," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 111–118.

[15] D. Zhou, Z. Zhong, D. Zhang, L. Shen, and C. Yan, "Autonomous landing of a helicopter uav with a ground-based multisensory fusion system," in *Seventh International Conference on Machine Vision (ICMV 2014)*. International Society for Optics and Photonics, 2015, pp. 94 451R–94 451R.

[16] W. Kong, T. Hu, D. Zhang, L. Shen, and J. Zhang, "Localization framework for real-time uav autonomous landing: An on-ground deployed visual approach," *Sensors*, vol. 17, no. 6, p. 1437, 2017.

[17] Y. Gui, P. Guo, H. Zhang, Z. Lei, X. Zhou, J. Du, and Q. Yu, "Airborne vision-based navigation method for uav accuracy landing using infrared lamps," *Journal of Intelligent & Robotic Systems*, vol. 72, no. 2, p. 197, 2013.

[18] D. Tang, T. Hu, L. Shen, D. Zhang, W. Kong, and K. H. Low, "Ground stereo vision-based navigation for autonomous take-off and landing of uavs: a chan-vese model approach," *International Journal of Advanced Robotic Systems*, vol. 13, no. 2, p. 67, 2016.

[19] S. Lange, N. Sunderhauf, and P. Protzel, "A vision based onboard approach for landing and position control of an autonomous multirotor uav in gps-denied environments," in *Advanced Robotics, 2009. ICAR 2009. International Conference on*. IEEE, 2009, pp. 1–6.

[20] A. Benini, M. J. Rutherford, and K. P. Valavanis, "Real-time, gpu-based pose estimation of a uav for autonomous takeoff and landing," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 3463–3470.

[21] S. Lin, M. A. Garratt, and A. J. Lambert, "Monocular vision-based real-time target recognition and tracking for autonomously landing an uav in a cluttered shipboard environment," *Autonomous Robots*, vol. 41, no. 4, pp. 881–901, 2017.

[22] F. Davide, Z. Alessio, S. Alessandro, D. Jeffrey, and D. Scaramuzza, "Vision-based autonomous quadrotor landing on a moving platform," in *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2017.

[23] D. Lee, T. Ryan, and H. J. Kim, "Autonomous landing of a vtol uav on a moving platform using image-based visual servoing," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 971–976.

[24] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[25] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[26] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks." in *Aistats*, vol. 15, no. 106, 2011, p. 275.

[27] P. WawrzyńSki and A. K. Tanwani, "Autonomous reinforcement learning with experience replay," *Neural Networks*, vol. 41, pp. 156–167, 2013.

[28] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[29] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," in *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993.

[30] K. Narasimhan, T. Kulkarni, and R. Barzilay, "Language understanding for text-based games using deep reinforcement learning," *arXiv preprint arXiv:1506.08941*, 2015.

[31] J. Engel, J. Sturm, and D. Cremers, "Camera-based navigation of a low-cost quadrocopter," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 2815–2821.

[32] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*.

[33] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks." in *Aistats*, vol. 9, 2010, pp. 249–256.

[34] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[35] R. Polvara, S. Sharma, J. Wan, A. Manning, and R. Sutton, "Vision-based autonomous landing of a quadrotor on the perturbed deck of an unmanned surface vehicle," *Drones*, vol. 2, no. 2, p. 15, 2018.

[36] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *arXiv preprint arXiv:1703.06907*, 2017.