Inclined Quadrotor Landing using Deep Reinforcement Learning*

Jacob E. Kooi[1] and Robert Babuška[2]

***Abstract*— Landing a quadrotor on an inclined surface is a challenging manoeuvre. The final state of any inclined landing trajectory is not an equilibrium, which precludes the use of most conventional control methods. We propose a deep reinforcement learning approach to design an autonomous landing controller for inclined surfaces. Using the proximal policy optimization (PPO) algorithm with sparse rewards and a tailored curriculum learning approach, a robust policy can be trained in simulation in less than 90 minutes on a standard laptop. The policy then directly runs on a real Crazyflie 2.1 quadrotor and successfully performs real inclined landings in a flying arena. A single policy evaluation takes approximately 2.5 ms, which makes it suitable for a future embedded implementation on the quadrotor.**

I. Introduction

Modern quadrotors are agile and can perform complex tasks in difficult-to-reach places. Quadrotor flight and manoeuvres are commonly controlled by proportional integral derivative (PID) control or model predictive control (MPC). Although these methods are adequate for setpoint or trajectory tracking, they fall short when it comes to more complicated manoeuvres that exceed the linearization range or require long prediction horizons. One of such manoeuvres is the landing on an inclined surface, which is relevant for applications like delivery, maintenance or surveillance. In order to facilitate a safe inclined landing, the final attitude of the quadrotor must match the slope of the landing platform. The final state of the landing trajectory is not an equilibrium, which presents a challenge for the control design. Owing to the under-actuated nature of the system, the landing trajectory can be long and complex, with an initial motion away from the landing location. This complicates the use of standard control methods like MPC with a fixed prediction horizon and quadratic cost function.

Recent advances in deep reinforcement learning (DRL) with continuous action spaces have made this approach suitable also for quadrotor control [1], [2], [3], [4], including landing controllers [5], [6], [7], [8], [9], [10]. However, no results have yet been reported for inclined landing. In this paper, we develop a DRL approach to the inclined landing problem and validate it in simulations and real lab experiments with the Crazyflie 2.1 Nano-UAV. To the best of our knowledge, this is the first deep-learning-based controller for inclined landing applied to a real quadcopter. More specifically, our contributions are:

- We develop two fast Gym-based [11] simulation environments for the Crazyflie 2.1 Nano-UAV.[3] One three-dimensional environment can be used with any compatible DRL algorithm to train set-point tracking. The other two-dimensional environment, restricted to the vertical xz-plane, can be used with an on-policy algorithm to train inclined landing. The resulting policies adequately transfer to the real Crazyflie.
- Building upon the state-of-the-art model-free proximal policy optimization (PPO) algorithm [12], we propose a powerful curriculum learning [13] approach to speed up the convergence when using sparse rewards, without the need for applying a multi-goal setting like in hindsight experience replay [14] or iterated supervised learning [15].
- We test the trained policy network in simulations and then deploy it onto the real Crazyflie quadrotor to demonstrate the actual inclined landing in an indoor flying arena.[4]

The remainder of the paper is structured as follows. We first give an overview of the related work in Section II. The dynamic quadrotor model used for simulation and training is described in Section III. Next, Section IV presents the DRL simulation framework that is used to train inclined landing and set-point tracking. Section V describes the simulation and lab setup and presents the experimental results. Finally, in Section VI, the conclusions and limitations of this work are given, along with proposals for future work.

II. Related Work

Deep reinforcement learning methods have been applied to a variety of quadrotor control problems, including hovering [3], attitude control [2], set-point tracking and disturbance recovery [1], [4]. Specifically for landing, a deep neural network was employed to learn higher-order interactions to stabilize the near-ground behavior of a nonlinear quadrotor controller [5]. A deep Q-learning network (DQN) was used to detect a marker symbol and perform a landing by using a downward-facing low-resolution camera [7], [8]. Least-squares policy iteration (LSPI) was employed to autonomously land on a marker [10] and the deep deterministic policy gradient (DDPG) algorithm [16] was used to navigate a descending quadrotor to land on a moving platform [9]. Finally, the work in [6] involved a convolutional

[1]Jacob E. Kooi is with the Departments of Cognitive Robotics and Delft Center for Systems and Control, Delft University of Technology, 2628 CD Delft, The Netherlands jacobkooi92@gmail.com

[2]Robert Babuška with the Department of Cognitive Robotics, Delft University of Technology, 2628 CD Delft, The Netherlands r.babuska@tudelft.nl

[3]https://github.com/Jacobkooi/InclinedDroneLander.git

[4]https://youtu.be/53YaqfwUIFU

neural network to estimate the heading angle to aid UAV landing in the case of a sensor failure. However, none of the approaches considered inclined landing and none of the methods developed can be directly applied to this problem.

Inclined landing has been the topic of several works outside the deep learning control literature. A nonlinear hybrid controller was proposed in [17]. A trajectory-tracking controller first guides the quadrotor above the landing platform and then switches to an attitude-tracking controller to ensure that the attitude of the quadrotor adjusts to the slope of the landing platform upon touchdown. This is an ad hoc local strategy, incapable of generating optimal landing trajectories from arbitrary initial conditions. Besides, no real-time control experiments have been reported in this paper. The method proposed in [18] features a nonlinear MPC to land a quadrotor on a moving inclined surface. Real-time experimental results were reported, showing a successful landing. The limitations of MPC are its computational complexity and the difficulty of parameter tuning, especially of the prediction horizon, which needs to be long for some of the landing trajectories, making the method unsuitable for embedded implementation on the quadrotor. The approach developed in [19] relies on splitting up the problem in the generation of dynamically feasible trajectories, and their subsequent trajectory tracking. Perching to slopes of up to 90 degrees has been demonstrated in lab experiments. To keep the problem tractable, the authors break the desired trajectory down into segments with a maximum duration of one second. The overall approach is more complex than the nonlinear feedback policy approach pursued in this paper.

III. Simulation Model

The dynamic model of the Crazyflie 2.1 Nano-UAV is formed by the equations of motion (EOM). We divide them into the Newton-Euler equations, which govern the axial accelerations, and an approximation of the body attitude control loops. The command input vector u to the Crazyflie's onboard controller is defined as

$$u = \begin{bmatrix} \Theta_c & \phi_c & \theta_c & \dot{\psi}_c \end{bmatrix}^T. \tag{1}$$

Here, Θ_c is the commanded pulse-width modulation (PWM) signal representing the total thrust, ϕ_c and θ_c are the commanded pitch and roll angles, respectively, and $\dot{\psi}_c$ is the commanded yaw rate [20]. These inputs are bounded by

$$\begin{aligned} u_{\min} &= \begin{bmatrix} 10000 & -30^o & -30^o & -200^o/s \end{bmatrix}^T, \\ u_{\max} &= \begin{bmatrix} 60000 & 30^o & 30^o & 200^o/s \end{bmatrix}^T. \end{aligned} \tag{2}$$

A. Newton-Euler Equations

The quadrotor is modeled as a rigid body, with the axial accelerations in the inertial frame $\begin{bmatrix} x & y & z \end{bmatrix}^T$:

$$\begin{bmatrix} m\ddot{x} \\ m\ddot{y} \\ m\ddot{z} \end{bmatrix} = \mathbf{R} \left(\begin{bmatrix} 0 \\ 0 \\ F_t \end{bmatrix} + F_a \right) + \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \tag{3}$$

with m the quadrotor's mass, $\mathbf{R}$ the rotation matrix from

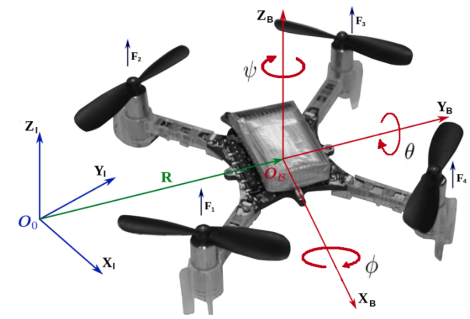

Fig. 1. The quadrotor coordinate system used throughout this paper. Subscripts B and I represent the body and inertial frame, respectively, and F_i is the thrust due to rotor i. Adopted from [21].

the body frame to the inertial frame, F_t the total thrust force and F_a the drag force. The rotation matrix corresponding to the axis frame representation in Fig. 1 is

$$\mathbf{R} = \begin{bmatrix} c\psi c\theta - s\phi s\psi s\theta & -c\phi s\psi & c\psi s\theta + c\theta s\phi s\psi \\ c\theta s\psi + c\psi s\phi s\theta & c\phi c\theta & s\psi s\theta + c\psi c\theta s\phi \\ -c\phi s\theta & s\phi & c\phi c\theta \end{bmatrix} \tag{4}$$

where c is a cosine, s is a sine, and ϕ, θ and ψ are the roll, pitch and yaw angles, respectively. The relation between the commanded PWM Θ_c in (1) and F_t in (3) is modelled by using the discrete-time transfer function found in [22] for an individual motor i:

$$\frac{F_i(z)}{\Theta_{c,i}(z)} = \frac{7.2345374 \cdot 10^{-8}}{1 - 0.9695404 z^{-1}} \qquad (500Hz). \tag{5}$$

Since the Crazyflie's onboard controller only takes a single PWM signal for all four motors, we assume that $F_t \approx 4F_i$ with $\Theta_c \approx \Theta_{c,i}$. Multiplying (5) by four and digitally converting it to continuous-time gives

$$\begin{bmatrix} \dot{\Omega} \\ F_t \end{bmatrix} = \begin{bmatrix} -15.467 \\ 1.425 \cdot 10^{-4} \end{bmatrix} \Omega + \begin{bmatrix} 1 \\ 2.894 \cdot 10^{-7} \end{bmatrix} \Theta_c \tag{6}$$

where Ω is an unmeasured state used for simulation purposes only. The drag force F_a in (3) is expressed as [23]

$$F_a = \boldsymbol{K}_a \omega_\Sigma v \tag{7}$$

with ω_Σ the sum of the rotor velocities, v the body-frame velocity vector and $\boldsymbol{K}_a$ a diagonal matrix of drag constants estimated in [22]. Because ω_Σ is not known during simulation, we approximate it from F_t with additional conversion formula's given in [22].

B. Body Attitude Control Loops

The body attitude rates are modelled by equations that approximate the dynamics of the attitude control loops [24]:

$$\begin{aligned} \dot{\phi} &= \frac{1}{\tau_\phi}(k_\phi \phi_c - \phi), \\ \dot{\theta} &= \frac{1}{\tau_\theta}(k_\theta \theta_c - \theta), \\ \dot{\psi} &= \dot{\psi}_c. \end{aligned} \tag{8}$$

Here τ_ϕ, τ_θ and k_ϕ, k_θ are the time and gain constants for roll and pitch, respectively. The yaw rate is assumed to instantaneously track the desired yaw rate, which is a reasonable assumption since yaw has no effect on the quadrotor's position [24]. Because the closed-loop dynamics are unknown, the parameters k_θ, k_ϕ, τ_θ and τ_ϕ need to be identified. Given the quadrotor's symmetry, k_θ and k_ϕ as well as τ_θ and τ_ϕ are assumed equal. These parameters are estimated by fitting data gathered by a motion-capture system to the equations (8). We conducted 20 experiments using square and sine waves ranging from zero to thirty degrees, which gave an average fit of 85.3% using Matlab's `nlgreyest` function, with the resulting parameters $k_\phi = k_\theta = 1.1094$ and $\tau_\phi = \tau_\theta = 0.1838\,\mathrm{s}$.

To simulate the quadrotor, the model equations (3) and (8) are integrated by using the fourth-order Runge Kutta (RK4) method. The step size is fixed and equal to the sampling period $T_s = 0.02\,\mathrm{s}$.

IV. Training Deep Reinforcement Learning Policies

To train the inclined landing, the quadrotor model of Section III is used as a simulation environment for model-free DRL. Additionally, to navigate the quadrotor, we train set-point tracking in the same fashion. Both policy networks map quadrotor states to the desired control input in (1), which makes them directly applicable to the real Crazyflie.

A. Preliminaries

The learning controller (agent) interacts with the model (environment) through trials. The environment's state-space is denoted by $\mathcal{S}$ and a specific value of the state at time step k by s_k. The agent applies an action $a_k \in \mathcal{A}$ and subsequently receives a reward $r_k \in \mathbb{R}$, after which it observes the next state s_{k+1}. The action a_k is chosen by following a stochastic policy $\pi_k(a|s)$ or a deterministic policy $\mu_k(s)$. This policy can be optimized in many different ways. Most techniques maximize the discounted return $\eta(\pi_\phi) = \mathbb{E}_\tau[\sum_{t=0}^{T} \gamma^t r(s_k, a_k)]$, with τ a trajectory following the policy π_ϕ.

B. Set-Point Tracking

We first train a three-dimensional set-point tracking policy network to empirically check the simulation-to-reality performance of the DRL algorithms and to fly to a desired starting position for inclined landing. For this task, the states and actions are defined as follows:

$$\begin{aligned} s_{3d} &= \begin{bmatrix} x & y & z & v_x & v_y & v_z & \phi & \theta \end{bmatrix}^T, \\ a_{3d} &= \begin{bmatrix} \Theta_c & \phi_c & \theta_c \end{bmatrix}^T. \end{aligned} \tag{9}$$

Here, v is the velocity in the inertial frame. Note that the yaw angle is kept constant at zero degrees, and can thus be omitted throughout all our experiments. For set-point tracking, we use the following reward function:

$$r_k = -e_p - 0.2e_v - 0.1e_{\phi,\theta} - 0.1\frac{a_{\phi,\theta}^2}{\max(e_p, 0.001)} \tag{10}$$

where e_p, e_v and $e_{\phi,\theta}$ are Euclidean distance errors of the position, velocity and orientation, respectively, with respect to the goal state. The term $a_{\phi,\theta}^2$ is the sum of the squared roll and pitch actions (normalized between 0 and 1). It is scaled by the reciprocal of e_p to minimize oscillations near the goal position.

The policy network is a fully connected neural network with two hidden layers, with 64 neurons each, and the tanh activation function everywhere except for the output layer which has a linear activation function. The final output is subsequently clipped between -1 and 1. We use the PPO algorithm [12] to train the set-point tracking network. Other state-of-the-art DRL algorithms like twin delayed deep deterministic policy gradient (TD3) [25] and soft actor-critic (SAC) [26] converged successfully as well, but PPO was superior in terms of the training time and the final policy performance.

C. Inclined Landing

The inclined landing is trained in the xz-plane, using the following states and actions:

$$\begin{aligned} s_{2d} &= \begin{bmatrix} x & z & v_x & v_z & \theta \end{bmatrix}^T, \\ a_{2d} &= \begin{bmatrix} \Theta_c & \theta_c \end{bmatrix}^T. \end{aligned} \tag{11}$$

For the sake of brevity, in the sequel, we refer to s_{2d} and a_{2d} by s and a, respectively. Because the quadrotor is under-actuated, an initial swinging motion away from the landing location is required for some initial conditions. This characteristic is incompatible with the bias a Euclidean distance based reward like the one in (10) generates. The reward function used for inclined landing is therefore a sparse reward defined as follows:

$$r_k = \begin{cases} 0 & \text{if} \quad s_k \in S_g \\ -\beta & \text{if} \quad s_k \in S_o \\ -2 & \text{if} \quad s_k \in S_b \\ -1 & \text{otherwise.} \end{cases} \tag{12}$$

Here $S_g = \{s \mid |s_i - s_{g,i}| < \delta_{g,i},\ \forall i\}$ is the set of goal states, defined as a hyperbox around the landing attitude. The goal threshold vector δ_g defines the desired landing tolerance and it is set by the user. The landing platform itself is an obstacle associated with a set of obstacle states S_o and a penalty β, and S_b represents the set of states close to the state space boundaries.

The use of a sparse reward requires extensive exploration in order to receive a non-negative reward and leads to prolonged training. We introduce the following curriculum learning [13] procedure to speed up the training:

- The training starts without a landing platform and with a horizontal goal state. Only once the quadrotor reliably reaches the horizontal goal, we begin slightly tilting the goal position after each episode. Finally, the landing platform is introduced into the environment, see Fig. 2.
- We initialize simulations near the goal state and with each episode expand the set of initial positions. This

eliminates the need for exploration by letting the non-negative rewards propagate throughout the value network at the beginning of training.
- We start with a large goal hyperbox S_g and as the training progresses, the hyperbox is gradually reduced to its desired size.

This learning curriculum requires an on-policy learning algorithm, such as PPO. Off-policy replay buffers would inevitably contain samples representing goals that are no longer relevant. In our experience, off-policy algorithms TD3 and SAC cannot keep up with the curriculum. The policy network architecture is similar to the one used for set-point tracking. The input and output layers for inclined landing are the state and action vectors in (11).

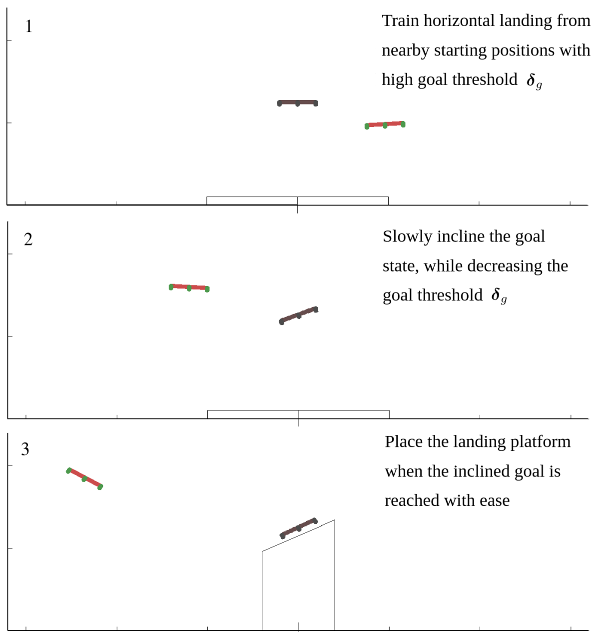

Fig. 2. The progress of the training curriculum for inclined landing. The red quadrotor represents the agent and the black quadrotor represents the goal state.

D. Simulation to Reality Transfer

To deploy the policies on the Crazyflie 2.1 Nano-UAV, the trained Pytorch network is converted to a Robot Operating System (ROS) node, which maps the state to the control input vector in (1). The positions and velocities come from a Kalman filter node, which appends the coordinates from the Optitrack motion capture system with the estimated inertial frame velocities. The quadrotor's orientation is taken from the Crazyflie's default onboard estimator. An overview of this process is given in Fig. 3.

V. Experiments

All simulations, DRL training and lab experiments are done on an HP Zbook Studio G4 laptop, with the default Nvidia Quadro M1200 GPU and an Intel Core i7-7700HQ CPU. The additional hardware used is the Optitrack motion-capture system, the standard Crazyflie 2.1 with a small

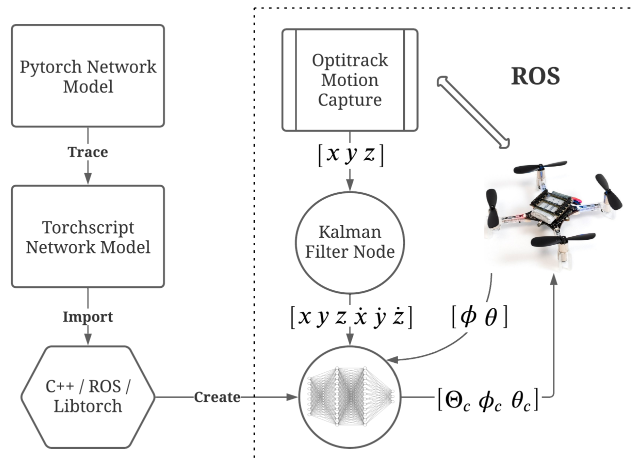

Fig. 3. Schematic overview of the Pytorch model deployment.

marker holder and a Crazyradio PA for communication with the Crazyflie.

A. Experimental Setup

The simulations and DRL training are implemented in Python, using the Pytorch package. The quadrotor dynamics are simulated using the EOM of Section III, integrated by the RK4 method. To increase the integration speed, we compile the EOM functions using Numba's Just-in-time (jit) package. By keeping the simulations and rendering in our own Gym-architecture [11] environment, we can use well-tuned implementations of existing model-free algorithms by [27]. The simulation bounds are equal to the dimensions of the actual flying arena with $\begin{bmatrix} x_{min} & x_{max} & y_{min} & y_{max} & z_{min} & z_{max} \end{bmatrix} = \begin{bmatrix} -3.4 & 3.4 & -1.4 & 1.4 & 0 & 2.4 \end{bmatrix}$ m. The actions represent the multiplication of the clipped policy network outputs $(a \in [-1, 1])$ by the control bounds in (2), with an exception for the PWM network output which is multiplied by 16500 and added to the estimated hover PWM of 42000. All simulations use the control sampling frequency of $50\,\text{Hz}$, with episode lengths of 300 time steps, i.e., 6 seconds.

1) Set-Point Simulations: The goal state is taken as $s_g = \begin{bmatrix} 0 & 0 & 1.2 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$, which represents the center of our physical flying arena. After each episode, the initial position is set randomly anywhere within the simulation bounds, with a small margin around the edges. The PPO implementation of [27] is used, with the discount factor γ reduced to 0.97 to account for more short-term control behaviour. We train for a total of 10^6 time steps representing 27 minutes of training time. For navigation with the resulting policy, a coordinate change suffices to fly the quadrotor to an arbitrary position.

2) Inclined Landing Simulations: For inclined landing, we operate the quadrotor in the xz-plane. The DRL algorithm used is the PPO implementation of [27], where minor changes have been made to activate rendering every 50 training iterations and to gradually start increasing γ from

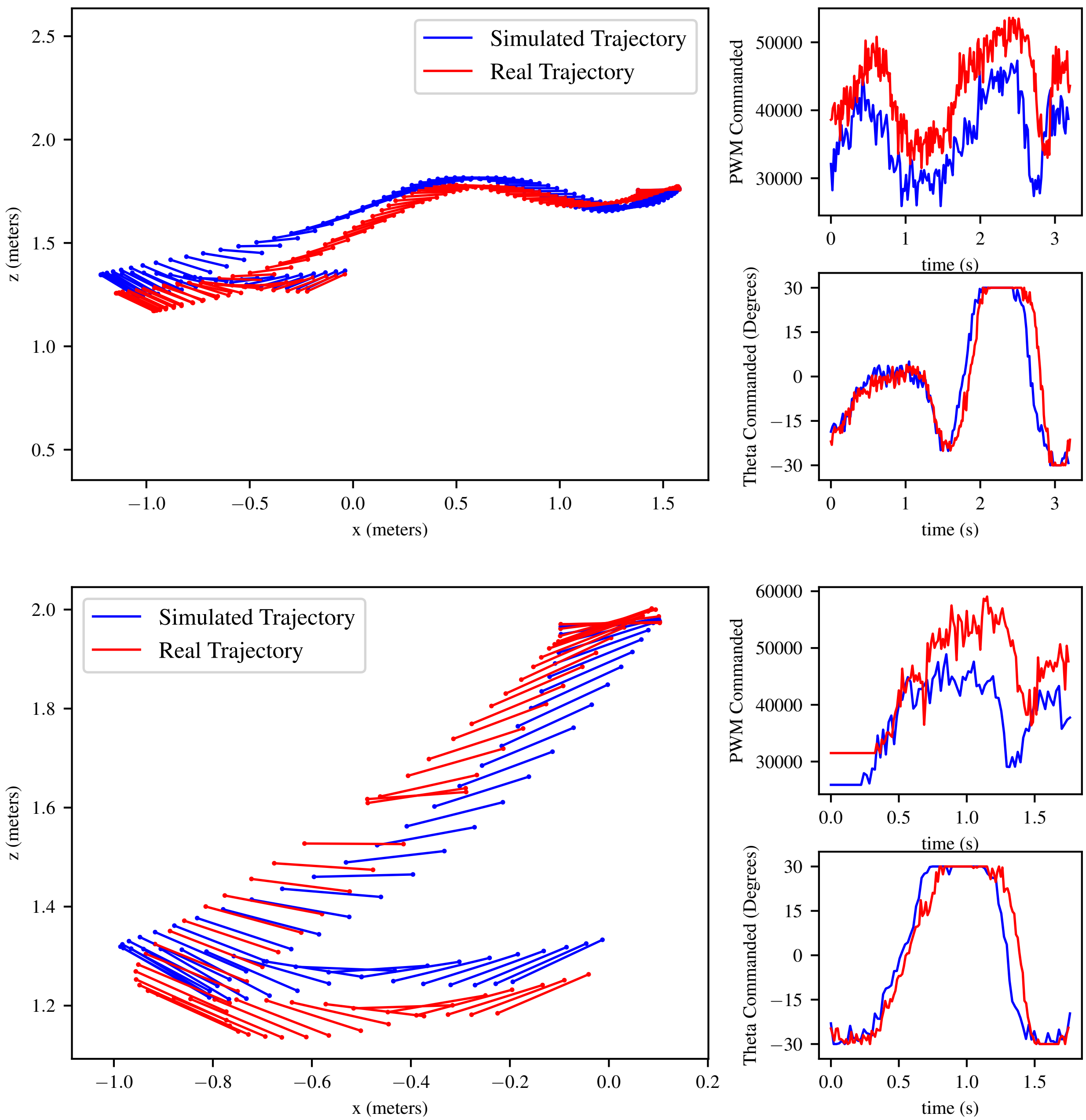

Fig. 4. Landing trajectories starting from above right (top) and from directly above (bottom) of the inclined goal position at (0, 1.25).

0.97 to 0.99 after 300 training iterations. An episode ends at 300 time steps or when the state x is within the goal hyperbox S_g. The landing platform is modeled as a polygon and appears after $8 \cdot 10^5$ time steps. The goal threshold vector is defined as $\delta_g = \begin{bmatrix}\delta_x & \delta_z & \delta_{v_x} & \delta_{v_z} & \delta_\theta\end{bmatrix} = \begin{bmatrix}d & d & min(10d, 1.5) & min(10d, 1.5) & 0.25d\end{bmatrix}$, where d starts at 0.25 in the beginning of training and gradually decreases to 0.10 after 2500 episodes with $0.15/5000$ per episode. The box of possible starting positions around the goal state expands with every episode by $1/6000\,\mathrm{m}$ in the x-direction and by $1/8000\,\mathrm{m}$ in the y-direction. Additionally, the goal state stays horizontal for the first $4 \cdot 10^5$ time steps and then gradually tilts towards its final inclination of $-\pi/7$ at the rate of $(-\pi/7)/6000$ radians per episode. The final goal state is set at $s_g = \begin{bmatrix}0 & 1.25 & 0 & 0 & -\pi/7\end{bmatrix}^T$. The obstacle reward constant β is taken as -7, which was found empirically to be the right trade-off between the goal of landing on top of the platform and the necessity to avoid the landing platform base. Training is stopped anywhere between $1.2 \cdot 10^6$ and $3 \cdot 10^6$ time steps (30 to 80 minutes), when the rendering shows that the policy executes the inclined landing reliably. Note that rather than monitoring the loss function, the aforementioned parameters and curricula have been empirically tuned by frequently rendering. The trend in the loss function value is quite meaningless, given the curricula and the discount factor adaptation.

3) Validation on a real quadrotor: The quadrotor used is the Crazyflie 2.1 Nano-UAV with its original firmware. A Crazyflie-specific package [20] allows us to publish the control values in (1) directly to the ROS server, which are then transmitted with low latency to the Crazyflie over the Crazyradio PA. The trained policy network is evaluated at $80\,\mathrm{Hz}$, even though it was trained at $50\,\mathrm{Hz}$. This is possible, as the policy is a function of the physical states only, and can therefore be evaluated at any frequency. A single policy evaluation takes approximately $2.5\,\mathrm{ms}$. The coordinates from the Optitrack motion-capture system come in at $120\,\mathrm{Hz}$ and are combined with the inertial frame velocities by the Kalman filter node [28]. The orientation is taken from the Crazyflie's onboard estimator, which is received through the Crazyradio at around 80Hz. The obtained position, velocity and orientation estimates form the policy input. The experiment itself starts with the drone flying to a position of choice, using the three-dimensional set-point tracking network. Once it is positioned, the networks are switched and the drone commences the inclined landing. This can begin from an arbitrary location in the top half of the flying arena. The landing trial ends when the quadrotor's state s_k is located in the set of goal states S_g.

B. Experiment Results

Early experiments showed a vertical offset between the simulated and real trajectories, caused by a slight inaccuracy of our motor thrust equation (6). We were able to compensate for this by increasing the hover PWM to 48000 during real testing. The resulting behaviour was very similar to the simulations, as can be seen in Fig. 4.[5] These trajectories originate from the same policy network, starting from arbitrary initial positions and ending when $s_k \in S_g$ with $\delta_g = \begin{bmatrix} 0.10 & 0.10 & 1.5 & 1.5 & 0.025 \end{bmatrix}$.

To further evaluate the performance, we measured the landing success rate when starting the flight from three different initial positions. Each position was evaluated 10 times, resulting in the success percentages reported in Table I. Even if an experiment did not succeed, the agent would not crash but autonomously fly back and forth, sometimes succeeding in its second or third attempt. However, these further attempts were not counted as a successful landing.

TABLE I

COMPARISON OF REAL-WORLD AND SIMULATION EXPERIMENTS

setting	success from initial (x, z)			**total**
	$(0, 2)$	$(-1.5, 1.6)$	$(1.5, 1.8)$	
Real-World	90%	70%	100%	**86.7%**
Simulation	90%	90%	100%	**93.3%**

No simulation-to-reality transfer techniques, such as domain randomization, have been employed, as they did not seem to improve final performance, while they did complicate the agent's training. Additionally, we found that using the Crazyflie's onboard orientation estimates rather than the Optitrack orientation estimates resulted in a substantial increase in performance.

The results show that inclined landing controllers can be designed by means of DRL. These controllers can transfer adequately to reality without the need for dynamics randomization or sensor noise. Furthermore, the same policy can be initiated from a wide variety of starting states. Results could further improve by additional system identification of the total motor thrust, where a slight mismatch might have been caused by making too strong assumptions about the scaling of the single motor model in (5) to the full motor model in (6). A perfect thrust model will however not exist, due to motor wear and tear and the strong relation of the drone's battery level to its thrust output, which is also reported in [1].

[5]Because of a faulty Optitrack measurement, the bottom figure shows a single misplaced red quad with a corresponding short drop in PWM output.

VI. CONCLUSION AND FUTURE WORK

We have presented a model-free DRL technique to facilitate autonomous quadrotor landing on an inclined surface. We trained a control agent with PPO, using sparse rewards and a learning curriculum. This allows the agent to gradually progress toward a more difficult task, i.e., larger inclination angles of the landing platform. Moreover, we have shown that the trained policies transfer well to reality, without employing any simulation-to-reality transfer techniques.

A limitation of this work is the fact that we restricted the landing trajectory to the xz-plane, which may cause some drift in the y-direction. A three-dimensional landing policy would increase precision, albeit at the cost of longer and more complex training. Furthermore, the reliance on external motion-capture systems makes this work hard to reproduce with onboard sensors only. An extension to such a setting is the topic of our future work. Additional future work could focus on larger inclination angles of the landing platform, as long as the onboard attitude controller would allow them. In this way, one could try to extend the findings in this paper to perching behavior similar to [19]. The goal inclination can also be added to the quadrotor's state, which would enable landing on unknown platforms, using, for instance, the laser measurement system of [17] or an onboard camera system like in [18]. Also, a form of the platform contact dynamics in [29] could be implemented in our simulator for a more robust landing and to aid the design of an end-to-end controller, which would eliminate the need for an external stopping signal.

ACKNOWLEDGMENT

Robert Babuška was supported by the European Union's H2020 project Open Deep Learning Toolkit for Robotics (OpenDR) under grant agreement No. 871449.

REFERENCES

[1] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, 06 2017.

[2] W. Koch, R. Mancuso, R. West, and A. Bestavros, “Reinforcement learning for uav attitude control,” *ACM Trans. Cyber-Phys. Syst.*, vol. 3, no. 2, Feb. 2019.

[3] N. O. Lambert, D. S. Drew, J. Yaconelli, R. Calandra, S. Levine, and K. S. J. Pister, “Low level control of a quadrotor with deep model-based reinforcement learning,” *IEEE Robotics and Automation Letters*, vol. 4, pp. 4224–4230, 2019.

[4] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, “Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors,” *arXiv preprint arXiv:1903.04628*, 2019.

[5] G. Shi, X. Shi, M. O’Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S. Chung, “Neural lander: Stable drone landing control using learned dynamics,” in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 9784–9790.

[6] Y. Bicer, M. Moghadam, C. Sahin, B. Eroglu, and N. K. Üre, “Vision-based uav guidance for autonomous landing with deep neural networks,” in *AIAA Scitech 2019 Forum*, 2019, p. 0140.

[7] 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,” in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018, pp. 115–123.

[8] R. Polvara, M. Patacchiola, M. Hanheide, and G. Neumann, “Sim-to-real quadrotor landing via sequential deep q-networks and domain randomization,” *Robotics*, vol. 9, no. 1, 2020.

[9] A. Rodríguez Ramos, C. Sampedro Pérez, H. Bavle, P. de la Puente, and P. Campoy, “A deep reinforcement learning strategy for uav autonomous landing on a moving platform,” *Journal of Intelligent & Robotic Systems*, vol. 93, 02 2019.

[10] M. B. Vankadari, K. Das, C. Shinde, and S. Kumar, “A reinforcement learning approach for autonomous control and landing of a quadrotor,” in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018, pp. 676–683.

[11] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[13] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML ’09. New York, NY, USA: Association for Computing Machinery, 2009, p. 41–48.

[14] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, “Hindsight experience replay,” in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[15] D. Ghosh, A. Gupta, A. Reddy, C. M. Devin, B. Eysenbach, and S. Levine, “Learning to reach goals via iterated supervised learning,” in *International Conference on Learning Representations*, 2021.

[16] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning.” in *ICLR*, 2016.

[17] J. Dougherty, D. Lee, and T. Lee, “Laser-based guidance of a quadrotor UAV for precise landing on an inclined surface,” *2014 American Control Conference*, pp. 1210–1215, 2014.

[18] P. Vlantis, P. Marantos, C. P. Bechlioulis, and K. J. Kyriakopoulos, “Quadrotor landing on an inclined platform of a moving ground vehicle,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 2202–2207.

[19] J. Thomas, M. Pope, G. Loianno, E. Hawkes, M. Estrada, H. Jiang, M. Cutkosky, and V. Kumar, “Aggressive flight for perching on inclined surfaces,” *Journal of Mechanisms and Robotics*, vol. 8, 2015.

[20] W. Hönig and N. Ayanian, *Flying Multiple UAVs Using ROS*. Springer International Publishing, 2017, pp. 83–118.

[21] Y. Chen and N. O. Pérez-Arancibia, “Nonlinear adaptive control of quadrotor multi-flipping maneuvers in the presence of time-varying torque latency,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–9.

[22] J. Förster and R. D’Andrea, “System identification of the crazyflie 2.0 nano quadrocopter,” Zürich, Tech. Rep., 2015.

[23] M. W. Mueller, M. Hamer, and R. D’Andrea, “Fusing ultra-wideband range measurements with accelerometers and rate gyroscopes for quadrocopter state estimation,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 1730–1736.

[24] M. Kamel, J. Alonso-Mora, R. Siegwart, and J. Nieto, “Robust collision avoidance for multiple micro aerial vehicles using nonlinear model predictive control,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 236–243.

[25] S. Fujimoto, H. van Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in *ICML*, 2018, pp. 1582–1591.

[26] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80, 10–15 Jul 2018, pp. 1861–1870.

[27] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, “Stable baselines 3,” https://github.com/DLR-RM/stable-baselines3, 2019.

[28] H. Zhu and J. Alonso-Mora, “Chance-Constrained Collision Avoidance for MAVs in Dynamic Environments,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 776–783, 2019.

[29] J. Bass and A. L. Desbiens, “Improving multirotor landing performance on inclined surfaces using reverse thrust,” *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5850–5857, 2020.