Regular Article

Learned Multiagent Real-Time Guidance with Applications to Quadrotor Runway Inspection

Kirk Hovell[1], Steve Ulrich[2] and Murat Bronz[3]
[1]PhD Candidate Carleton University Ottawa, Canada
[2]Associate Professor Carleton University Ottawa, Canada
[3]Assistant Professor ENAC, Université de Toulouse Toulouse, France

Abstract: Aircraft runways are periodically inspected for debris and damage. Instead of having pilots coordinate the motion of the quadrotors manually or hand-crafting the desired quadrotor behavior into a guidance law, this paper reports the use of deep reinforcement learning to learn a closed-loop multiagent real-time guidance strategy for quadrotors to autonomously perform such inspections. This yields a significant reduction in engineering effort while enabling highly-flexible real-time performance. The runway is discretized into a number of rectangular tiles, which must all be visited for the runway to be considered inspected. The guidance system reported here calculates a desired acceleration in real time for the quadrotor(s) to track in order to complete the task. This paper first develops the guidance technique, trains it in simulation, and evaluates it experimentally using an indoor quadrotor laboratory. This process is then repeated for an outdoor setting on a real runway, where the proposed guidance strategy is compared to a handcrafted strategy and applied to a multiquadrotor scenario where the quadrotors must learn to coordinate their behavior and be resilient to the failure of one quadrotor mid-experiment. Multiagent, fault-tolerant, learned behavior is successfully demonstrated through outdoor quadrotor flights. Additional simulations and experiments demonstrate the technique is viable in a swarm with additional quadrotors, on a variety of runway shapes and with increased discretization of the runway. This work shows how modern learning-based techniques can: 1) reduce the engineering effort required to design complex guidance systems and 2) be implemented on real hardware in a representative outdoor environment.

Keywords: deep reinforcement learning, multiagent, quadrotor inspection

1. Introduction

Airport runways are inspected multiple times per day to ensure no debris or damage exists that may pose a safety hazard to aircraft. Typically, takeoffs and landings are paused to allow for a ground vehicle to safely drive down the runway to perform a visual inspection. To minimize delays

Received: 17 June 2021; revised: 22 April 2022; accepted: 05 May 2022; published: 8 June 2022.

Correspondence: Kirk Hovell, PhD Candidate Carleton University Ottawa, Canada, Email: kirk.hovell@carleton.ca

DOI: https://doi.org/10.55417/fr.2022036

to customers, these runway shutdown durations should be minimized. Recently, quadrotors have been flown by pilots for this task (Absolon et al., 2015; Sappington et al., 2019). Quadrotors have become useful in recent years due to the increased availability and reduced costs of microcontrollers, batteries, and 3D printing. Common quadrotor applications are search and rescue, surveillance, package delivery, defense, sport racing, and research. This work proposes using multiple quadrotors to quickly and autonomously inspect a runway. The quadrotors could capture images of the runway which could be merged into a composite image of the entire runway and analyzed for debris, damage, and, over time, longer-term runway problems. The autonomous acquisition of such images must rely on an intelligent trajectory guidance and control system. A variety of quadrotor guidance and control theories have been developed for such purposes (Kendoul, 2012). Backstepping control (Madani and Benallegue, 2006), sliding mode control (Xu and Ozguner, 2006), and dynamic inversion techniques (Das et al., 2009; Smeur et al., 2016a; Smeur et al., 2016b; Smeur et al., 2018) have all been applied to quadrotor control. Guidance, also known as path planning, has been demonstrated for individual quadrotors (Hoffmann et al., 2008; Mellinger and Kumar, 2011; Mellinger et al., 2012) and swarms of quadrotors (Honig et al., 2018; Shah et al., 2020). Coverage path planning (CPP) algorithms are a class of algorithm developed to guide an object to fully explore a given area. Many single-agent CPP algorithms use simple geometric patterns such as spirals, sweeps, or more advanced techniques such as cell decomposition (Zelinsky et al., 1993; Oksanen and Visala, 2009) or optimal methods (Cabreira et al., 2019). An overview of multiagent CPP can be found in (Almadhoun et al., 2019). While the quadrotor flight is autonomous using these advanced guidance and control techniques, the techniques themselves are handcrafted by researchers and may require significant engineering effort. In addition, many handcrafted guidance strategies are computationally intensive and their trajectories must be generated offline for open-loop use. This work, in contrast, allows for autonomous multiagent flight to be learned rather than designed. Once the offline computationally intensive learning process is complete, the resulting guidance model can easily generate trajectories in real time.

Deep reinforcement learning is a branch of artificial intelligence that, through trial and error, discovers an appropriate behavior to take in a given circumstance and generalizes this behavior well to new experiences. The trial-and-error behaviors are given the notion of *good* or *bad* through a human-designed reward function, which may be very simple. For example, giving a reward of +1 for winning a chess game, a reward of -1 for losing a chess game, and a reward of 0 for every move during the game has led to the most powerful chess engine in existence (Silver et al., 2017a). The notion of delegating the learning to the reinforcement learning algorithm rather than hand-crafting the desired behaviors can lead to entities that outperform humans in a variety of tasks and games (Mnih et al., 2015; Silver et al., 2017a; Silver et al., 2017b; Vinyals et al., 2019). In principle, it removes the burden of the designer from 1) knowing how to solve the problem and 2) determining how to encode that behavior.

Training reinforcement learning algorithms on real robots is difficult due to the trial-and-error nature of the learning process. Even with state-of-the-art fast-learning algorithms (Yang et al., 2019; Haarnoja et al., 2018), a robot may require hundreds or thousands of attempts before the learning succeeds, which is expensive and may lead to significant wear and tear on the robot. An alternate method is to train the reinforcement learning algorithm in simulation and to deploy the trained model to a real robot. However, problems arise with this approach since the simulated environment cannot perfectly match the real environment. The learned model overfits the simulated dynamics and performs poorly in the real world; this is known as the *simulation-to-reality problem*. Efforts to get around this problem involve randomizing the environmental parameters during training, known as domain randomization (OpenAI, 2018; OpenAI et al., 2019; OpenAI et al., 2020; Lee et al., 2019; Peng et al., 2018; Loquercio et al., 2020; Sadeghi and Levine, 2017; Van Baar et al., 2019), or fine-tuning once deployed to experiment (Van Baar et al., 2019; Cutler and How, 2016; James et al., 2019).

Multiagent deep reinforcement learning has also been explored in previous work (Vinyals et al., 2019; Buşoniu et al., 2008; Tan, 1993). Typically, each agent present learns its own behavior

model. However, this causes the environment to become "nonstationary," i.e., for a given agent, the environment, which includes the other agents' behavior, changes with time and slows learning. Inspired by recent work (Terry et al., 2020; Gupta et al., 2017; Chu and Ye, 2017), this paper shares a single behavior model among all agents. As discussed in Sec. 2.2.3, this allows agents to implicitly share behavior knowledge while increasing the data collection rate and removing the nonstationarity from the environment, as proven by (Terry et al., 2020).

Due to the inherent advantages discussed above, reinforcement learning has been used in quadrotor trajectory guidance and control applications. It has been used to learn an inner-loop controller that outperformed classical control techniques (Koch et al., 2019; Wang et al., 2020), a stabilization control task that included a PD controller (Hwangbo et al., 2017), and, along with domain randomization, has been used to enable real quadrotor flight (Sadeghi and Levine, 2017; Loquercio et al., 2020). Other learning techniques have been used to guide quadrotors through a maze with a discrete set of nodes (Junell et al., 2015; Greatwood and Richards, 2019). Reinforcement learning was used to train a basic vision-based guidance system in experiment (Siddiquee et al., 2019; Camci and Kayacan, 2019). Lastly, a multiagent stochastic wildfire surveillance guidance technique was developed with simulated results (Julian and Kochenderfer, 2019).

Unless otherwise stated, the above applications used deep reinforcement learning to learn a combined guidance *and* control strategy. In other words, control efforts for the robot to execute were calculated directly from the system state—the intermediate guidance and control aspects were combined into a single end-to-end calculation. However, it is possible that the guidance portion (which calculates the trajectory) of the learned system is appropriate for solving the task but the control portion (which commands the actuators to track the trajectory) has overfit the simulated dynamics and reduces performance. In this context, and as suggested by (Harris et al., 2019), it may therefore be beneficial to restrict reinforcement learning to learn a guidance behavior only, with the control aspect being left to the well-developed control theory community.

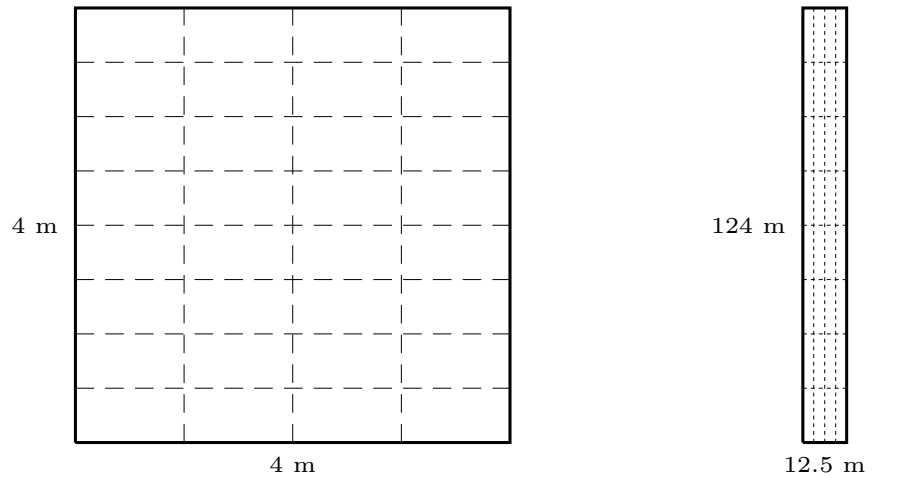
The authors' past work consisted of restricting the reinforcement learning algorithm to exclusively learn a guidance policy and letting an on-board controller handle any errors encountered while crossing the simulation-to-reality gap. It calculated closed-loop real-time velocity-based guidance signals for spacecraft proximity operations (Hovell and Ulrich, 2021) and was recently improved for use on quadrotor proximity operations where acceleration-based guidance signals were calculated (Hovell et al., 2021). This previous work considered single-agent scenarios only. The authors named the deep reinforcement learning-based guidance technique *deep guidance*.

In this work, the deep guidance technique is extended to a multiagent application and is used to generate real-time acceleration commands for multiple quadrotors to guide their inspection of a runway. Such a system is trained in simulation and tested in two experimental facilities at École Nationale de l'Aviation Civile in Toulouse, France. An indoor facility is first used for proof-of-concept experiments to develop the runway inspection technique. Then, outdoor GPS-driven real-world quadrotor runway inspection experiments are presented. The proposed guidance technique is also compared to a handcrafted technique. In addition, the fault tolerance of a two-quadrotor model is explored, the use of quadrotor swarms is studied, sensitivity to the runway grid size is explored, and the applicability of the technique to nonrectangular shapes is also presented.

The original contributions of this paper are as follows:

1. A novel, learned, multiagent, fault-tolerant runway inspection guidance algorithm.
2. The application and analysis of the technique to both indoor and outdoor quadrotor experimental facilities.

This paper is organized as follows: Section 2 discusses the multiquadrotor runway inspection scenario considered and how the deep guidance technique is applied to it, Sec. 3 discusses the experimental facilities present, Sec. 4 presents numerical simulations showing the training of the system and simulated performance, Sec. 5 presents experimental results in both the indoor and outdoor facilities, and Sec. 6 concludes this paper.



(a) Indoor runway **(b)** Outdoor runway

Figure 1. Runway shapes and inspection grids. Each tile must be passed over to be considered inspected. To solve the runway inspection task, all tiles must be inspected.

2. Runway inspection

This section discusses the multiquadrotor runway inspection environment within which the deep guidance system will be trained. It then presents the deep reinforcement learning implementation for the environment. In contrast to the authors' previous work, where the deep guidance technique was developed using simple tasks with shaped reward fields (Hovell et al., 2021; Hovell and Ulrich, 2021), the deep guidance technique is herein used to learn a task that is significantly more difficult and has a more abstract, sparse, reward function. The D4PG (Barth-Maron et al., 2018) reinforcement learning algorithm is used in this work, the details of which may be found in Appendix A.

2.1. Problem statement and modeling

The task is presented as follows: An arbitrary number of quadrotors, Q, must inspect a runway. The runway is divided into a number of tiles, as shown in Figure 1. Each quadrotor has full knowledge of its and the other quadrotors' positions and velocities. Each time a previously undiscovered runway tile is inspected, a reward of +1 is given to all quadrotors. A small penalty is given to quadrotors who become too close to each other to discourage collisions. Through attempting to maximize the rewards received, the learning algorithm will, in turn, learn an approach to cooperatively inspect the runway.

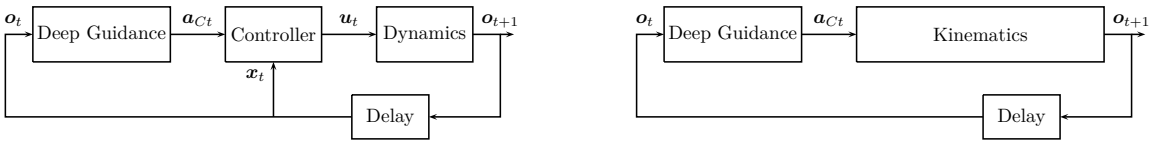
The position and velocity of the i^{th} quadrotor at time step t are

$$\boldsymbol{x}_t^i = \begin{bmatrix} x_t^i & y_t^i \end{bmatrix} \tag{1}$$

$$\boldsymbol{v}_t^i = \begin{bmatrix} v_{x_t}^i & v_{y_t}^i \end{bmatrix} \tag{2}$$

where x and y represent the positions in Cartesian space and v is the velocity. Only planar motion is considered. The matrix containing the state of the runway, where each element corresponds to a runway tile in Figure 1, has the form

$$\boldsymbol{S}_t = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1S_W} \\ s_{21} & s_{22} & \cdots & s_{2S_W} \\ \vdots & \vdots & \ddots & \vdots \\ s_{S_L1} & s_{S_L2} & \cdots & s_{S_LS_W} \end{bmatrix} \tag{3}$$



(a) Used in experiment and to evaluate simulations **(b)** Used during training

Figure 2. Block-scheme diagrams of the deep guidance strategies used in experiment (with a real controller) and in training (with an ideal controller assumed).

where $s_{jk} = \{0, 1\}$ represents the state of the j^{th} row and k^{th} column tile of the runway, and S_L and S_W represent the number of tiles in the length and width, respectively. A value of 0 represents the tile being uninspected, and a value of 1 indicates the tile has been inspected.

2.1.1. Deep guidance

The deep guidance technique allows for reinforcement learning to be used on real robot platforms, despite being trained entirely in simulation, by limiting deep reinforcement learning to only learn the guidance portion of the guidance, navigation, and control process (Hovell et al., 2021; Hovell and Ulrich, 2021). This constitutes the authors' solution to the simulation-to-reality gap. The learned closed-loop guidance system calculates an acceleration signal at each time step, and passes that acceleration to a conventional controller to track. The controller's ability to track the acceleration signal, regardless of modelling errors, is the motivation to restrict the reinforcement learning algorithm to learn guidance only.

Figure 2a shows a block-scheme diagram of the approach. The acceleration signal is $\boldsymbol{a}_{Ct}$ and the control effort is $\boldsymbol{u}_t$. To prevent the deep guidance policy from overfitting a particular controller during training, an ideal controller is assumed. The ideal controller and dynamics blocks can be combined into a single kinematics block, as shown in Figure 2b. Any controller may be used along with the deep guidance system to control a real robot, so long as it tracks the desired accelerations accurately.

2.1.2. Kinematics model

The deep guidance policy input observation for the i^{th} quadrotor is

$$\boldsymbol{o}_t^i = \begin{bmatrix} \boldsymbol{x}_t^i & \boldsymbol{v}_t^i & \boldsymbol{x}_t^{i+1} & \boldsymbol{v}_t^{i+1} & \cdots & \boldsymbol{x}_t^{\text{mod}(i+Q-1,\ i)} & \boldsymbol{v}_t^{\text{mod}(i+Q-1,\ i)} & \boldsymbol{e}^{\text{T}}(\boldsymbol{I} \otimes \boldsymbol{S}_t)^{\text{T}} \end{bmatrix}^{\text{T}} \tag{4}$$

for Q quadrotors, where $\boldsymbol{I}$ is an identity matrix of size $S_W \times S_W$, $\boldsymbol{e}$ is a column matrix of length S_W^2 filled with zeros except for ones in rows $(j-1)S_W + j \quad \forall\ j = 1, 2, \ldots, S_W$, and $\otimes$ is the Kronecker product; the final term reshapes the runway state matrix in Eq. (3) into a row vector. When this observation is passed through the deep guidance policy in Eq. (1), an action, which in this application is an acceleration signal, $\boldsymbol{a}_{Ct}^i$, is returned. During the training it is assumed that the ideal controller perfectly achieves the desired acceleration signal. Therefore, the guided acceleration signal is integrated twice to obtain the next state. All integration is performed using the SciPy (Oliphant, 2007) Adams/Backward differentiation formula methods in Python. When delays are included, as discussed in the following subsection, the guided acceleration is stored and the acceleration D time steps old is used instead.

2.1.3. Time delay consideration

Delays may arise from actuation delays, signal delays, or measurement delays. In the event that the desired acceleration is not immediately realized by the quadrotor, the observation no longer contains sufficient information for an appropriate action to be calculated—the Markov assumption has been violated. If the observation is "augmented" with past actions equal to the number of time steps of system delay present, the problem can be alleviated (Katsikopoulos and Engelbrecht, 2003). With

a delay of D time steps, the system's augmented observation becomes

$$\boldsymbol{o}_t^i = \begin{bmatrix} \boldsymbol{x}_t^i \\ \boldsymbol{v}_t^i \\ \boldsymbol{x}_t^{i+1} \\ \boldsymbol{v}_t^{i+1} \\ \cdots \\ \boldsymbol{x}_t^{\mathrm{mod}(i+Q-1,\ i)} \\ \boldsymbol{v}_t^{\mathrm{mod}(i+Q-1,\ i)} \\ \boldsymbol{a}_{Ct-1}^i \\ \boldsymbol{a}_{Ct-2}^i \\ \vdots \\ \boldsymbol{a}_{Ct-D}^i \\ (\boldsymbol{I} \otimes \boldsymbol{S}_t)\boldsymbol{e} \end{bmatrix} \tag{5}$$

Although state augmentation allows for optimal policies to be found despite system delays, if the delay is too long or too many quadrotors are used, the observation may grow too large for learning to be tractable, as explored in Secs. 4.3 and 4.4.

2.1.4. Dynamics model

To prevent the deep guidance policy from overfitting any simulated dynamics or controller, it is trained within the kinematics environment presented in Sec. 2.1.2. Occasionally, however, its performance must be evaluated in a dynamics environment with a controller in much the same way that it will be evaluated once deployed to a real quadrotor experiment. Figure 2a shows the type of simulation the policy is tested within. A planar double-integrator dynamic model is used:

$$\ddot{x}^i = \frac{u_x^i}{m^i} \tag{6}$$

$$\ddot{y}^i = \frac{u_y^i}{m^i} \tag{7}$$

where u_x^i and u_y^i are the forces applied to the i^{th} quadrotor along the X and Y axes, m^i is its mass, and $\ddot{x}^i$ and $\ddot{y}^i$ are its accelerations in X and Y, respectively. The accelerations are numerically integrated twice to obtain the position at the following time step. Note that only planar X and Y translational motion is considered since altitude changes are not needed for runway inspection.

An integral controller of the form

$$\boldsymbol{u}_t^i = \boldsymbol{u}_{t-1}^i + \boldsymbol{K}_I \left(\boldsymbol{a}_{Ct}^i - \dot{\boldsymbol{v}}_t^i\right) \tag{8}$$

is used in simulation, with $\boldsymbol{K}_I$ as an integral gain matrix, chosen by trial and error, and $\boldsymbol{u} = [u_x \quad u_y]^{\mathrm{T}}$. Each quadrotor's controller outputs a control effort that is executed on its simulated dynamics. The kinematics model used for training and the dynamics model used for evaluation are vastly simpler than the true dynamics of a quadrotor. This large discrepancy highlights the ability of the deep guidance technique to handle differences between simulation and reality.

2.1.5. Inspecting nonrectangular areas

While this paper was motivated by the runway inspection task, this subsection considers the applicability of the technique to other nonrectangular inspection tasks. For example, agricultural inspection or search and rescue may require nonrectangular areas to be inspected. Two shapes are considered: a C-shaped area and an L-shaped area, shown in Figure 3.

The nonrectangular areas use an identical logic: all tiles within the inspection area boundary yield a reward of +1. Therefore, the only modification needed for nonrectangular shapes is to change the structure of $\boldsymbol{S}_t$ in Eq. (3). All other training parameters are identical.

(a) C-shaped area **(b)** L-shaped area

Figure 3. Nonrectangular areas to inspect. Each tile must be passed over to be considered inspected. To solve the runway inspection task, all tiles must be inspected.

2.2. Methods

With the problem and modeling defined, the following sections describe the methods used to solve the runway inspection using deep reinforcement learning.

2.2.1. Reward function

The quadrotors are tasked with autonomously and cooperatively learning how to inspect an aircraft runway. The following logic is used to convey the desired behavior through a scalar reward returned to the quadrotors at each time step:

1. The runway is divided into a number of rectangular tiles in a grid that is S_L tiles long and S_W tiles wide, as shown in Figure 1.
2. If any quadrotor visits a new grid tile that was previously uninspected, a reward of $+1$ is given to all quadrotors.
3. If a quadrotor visits a previously inspected tile, a reward of 0 is given.
4. A penalty is given to individual quadrotors who become too close to another quadrotor in order to encourage independent inspection and discourage collisions. If there are many quadrotors, the proximity to only the nearest quadrotor is used.

The runway state, $\boldsymbol{S}_t$, is initialized as a grid of zeros. When a tile becomes inspected, it is replaced with a 1. The above logic is written as a scalar reward function for the i^{th} quadrotor at time step t through

$$r_t^i = \mathrm{Tr}\{\boldsymbol{S}_t^{\mathrm{T}}\boldsymbol{S}_t\} - \mathrm{Tr}\{\boldsymbol{S}_{t-1}^{\mathrm{T}}\boldsymbol{S}_{t-1}\} - c_1 e^{-\frac{\min_j \|\boldsymbol{x}_t^i - \boldsymbol{x}_t^j\|}{c_2}} \quad \forall\, j \in Q \wedge j \neq i \tag{9}$$

for some positive scaling constants c_1 and c_2, and where $\mathrm{Tr}\{\cdot\}$ represents the trace operation. The first two terms calculate the change in the runway's state while the third term penalizes quadrotor proximity.

An investigation is also performed in Sec. 5.3 into whether additional penalties for quadrotors leaving the runway discourages the behavior. In that test only, an additional penalty of 0.05 per time step is given to any quadrotor that leaves the runway.

2.2.2. Learning algorithm implementation details

The policy and value neural networks have the same shape: 400 neurons in their first hidden layer and 300 neurons in their second—Figure 19 is not to scale. For the value network, the action input

skips the first hidden layer, as empirically this was shown to yield better results (Lillicrap et al., 2016; Barth-Maron et al., 2018). Neurons in all hidden layers use the rectified linear unit (ReLU) as their nonlinear activation function, shown below:

$$g(y) = \begin{cases} 0 & \text{for } y < 0 \\ y & \text{for } y \geq 0 \end{cases} \tag{10}$$

The output layer of the policy network uses a $g(y) = \tanh(y)$ activation function to ensure the output is bounded and is then scaled to the action range, and the output layer of the value network uses a softmax function to ensure it outputs a valid probability distribution,

$$g(y_j) = \frac{e^{y_j}}{\sum_{k=1}^{B} e^{y_k}} \qquad \forall\ j = 1, \ldots, B \tag{11}$$

for each neuron y_j and for the number of bins in the distribution, B. As per the original value distribution paper, $B = 51$ (Bellemare et al., 2017). The bins are divided evenly across the expected range of total rewards received during an episode: $[0, S_L S_W]$. The Adam (Kingma and Ba, 2015) stochastic gradient descent algorithm is used to train the neural networks, with learning rates of $\alpha = \beta = 0.0001$. To avoid the vanishing gradients problem (Hochreiter, 1998), the inputs to the neural networks are normalized. Minibatches of $M = 256$ data points are randomly drawn from the replay buffer R, of size 10^6 samples, to train the networks. A discount factor of $\gamma = 0.99$ and an N-step return of $N = 5$ are used. The parameters for the smoothed policy and value networks are updated on each training iteration with $\epsilon = 0.001$. All parameters were tuned using trial and error with the goal of speeding up learning. To force the $K = 10$ actors to inspect their environment, noise is applied to their chosen actions with a standard deviation of $\sigma = \frac{1}{3}\left[\max(\boldsymbol{a}) - \min(\boldsymbol{a})\right](0.99998)^E$, where E is the episode number. This empirically leads to good inspection of the action space, and the decaying noise refines the search space as learning progresses. A dynamics delay of length $D = 3$ is used, as this was found to be the delay present in both the indoor and outdoor experimental facility when the time step was 0.2 s. The Tensorflow[1] machine learning framework is used to generate, train, and evaluate the neural networks.

After a designated agent performs five training episodes, the most up-to-date policy is "deployed" and run in a full dynamics environment with a controller, as described in Sec. 2.1.4. During deployment, $\sigma = 0$ in Eq. (10) such that no exploration noise is applied and the performance can be readily evaluated. All code used can be found at https://github.com/Kirkados/Field_Robotics_2021.

2.2.3. Multiagent considerations

Coordinating the behavior of multiple quadrotors requires careful consideration due to the nonstationarity of the environment. A nonstationary environment changes with time, which makes learning more difficult since the policy's knowledge of the environment becomes invalid over time. If each agent present learns a separate policy, nonstationarity emerges since each agent attempts to learn the best actions according to the environment it experiences—the other agents are therefore viewed as part of the environment. As an example, Agent 1 will learn a behavior based on its environment, which includes Agent 2's behavior. Then Agent 2 will learn a behavior based on its environment, which includes Agent 1's new behavior. Even though the environment is not changing, the changing agents cause information to "ring" between agents and leads to slow learning (Terry et al., 2020).

Instead of learning a separate policy for each agent, methods for sharing one policy between agents exist (Gupta et al., 2017; Chu and Ye, 2017). The options are: a) just the policy can be shared; b) just the value network can be shared; or c) both the policy and value networks can be shared. Option c) is used in this paper, where a single policy and value network are shared among all agents. Through each agent using the same policy and value network, an understanding for the other

[1] Software available from https://www.tensorflow.org

agents' behaviors is implicitly communicated. This removes the nonstationarity of the environment and increases the learning speed as proven in (Terry et al., 2020).

The observation of the system in Eq. (5) contains the positions and velocities of all quadrotors. When calculating the acceleration for quadrotor i, its own position $\boldsymbol{x}_t^i$ and velocity $\boldsymbol{v}_t^i$ are the first two entries in the observation, followed by the other quadrotors' positions and velocities, followed by the i^{th} quadrotor's past actions (discussed in Sec. 2.1.3), followed by the runway state $\boldsymbol{S}_t$ flattened into a column vector. By positioning a quadrotor's own information in the observation first, the policy is hypothesized to learn patterns about how to coordinate the motion of that quadrotor given the knowledge of the others. The same policy can therefore be used for all quadrotors—the observation is simply tailored to each quadrotor. A downside to including all quadrotors' state information in the observation is that the problem does not scale well. As more quadrotors are added, the observation grows and will eventually become intractable; this is explored in Sec. 4.4.

Through sharing the policy and value networks among all agents, the total number of neural networks needed is reduced and the data collection rate is increased. At each time step, all Q quadrotors take actions and experience the environment. Therefore, each quadrotor's experience can be separately assembled into an observation and logged in the replay buffer R along with its action, reward, and next observation. Therefore, each of the K parallel simulations generate Q data points at each time step.

2.2.4. Mission-level fault tolerance

To further increase the flexibility and reliability of the multiagent deep guidance approach, fault tolerance was built into the deep guidance model. In other words, a single model should complete the runway inspection task with two quadrotors. However, if one quadrotor fails, the same model should automatically command the remaining quadrotor to complete the runway inspection. Since all trajectories are generated in real time (through the acceleration signals), the surviving quadrotor's trajectory can be automatically modified upon a quadrotor failure.

Fault tolerance in this context refers to any event that prevents one quadrotor from completing its mission and may include IMU failures, GPS failures, low battery, communication loss, rotor failures, or major wind. Regardless of the failure sources, which are outside the scope of this work, they can be identified by the loss of communication or quadrotor motion. Once the failure is identified, the surviving quadrotor shall continue the runway inspection and complete the task.

To build in this fault tolerance, it must be experienced during training. During the training of the dual-quadrotor model, one quadrotor is programmed to fail 50% of the time within the first 30 s of the episode. When a quadrotor fails, its position and velocity are fixed at a certain value and its collected data are no longer used for training purposes. As training progresses, the still-operational quadrotor will learn to recognize when the other quadrotor has failed. The result is a dual-quadrotor deep guidance model that can tolerate quadrotor failures automatically. This fault tolerance is in addition to the other benefits of the deep guidance approach: flexibility in initial condition, real-time trajectory generation, and automatic cooperation between quadrotors—all of which are learned. When the outdoor dual-quadrotor model is evaluated, both in simulation (Sec. 4.2) and experiment (Sec. 5.3), the ability of the policy to both inspect the runway with and without a quadrotor failure is presented. In Sec. 5.3 the tolerance to a quadrotor failure when it was not experienced during training is also tested.

2.2.5. Handcrafted zigzag approach

The deep guidance runway inspection strategy discussed here is compared to a handcrafted approach. An intuitive zigzag strategy to inspect the runway is shown in Figure 4. This handcrafted set of waypoints readily guides the quadrotor to inspect all tiles. The simplicity of this approach makes it inflexible to changing initial conditions—the quadrotor will need to waste time flying to the starting point from its initial condition; however, it is similar in design effort to the deep guidance system reward function used in this work (a reward of +1 for inspecting a new runway tile; penalties for quadrotor proximity). These two methods are compared while attempting to hold their design

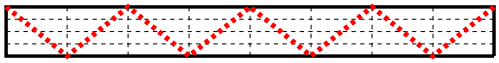
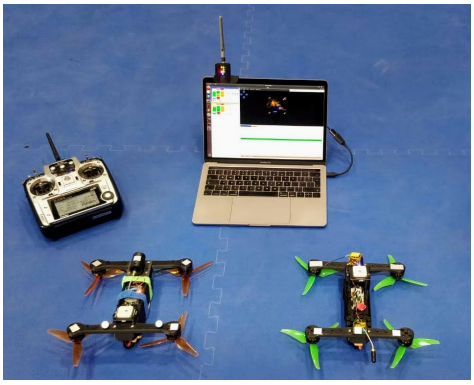


Figure 4. The handcrafted zigzag approach. The red trajectory represents the waypoints used to guide a quadrotor to reliably inspect the outdoor runway.

(a) Indoor flight arena



(b) Ground computer and quadrotors

Figure 5. ENAC indoor facility and flight hardware.

effort equal. Advanced coverage path planning algorithms exist (Zelinsky et al., 1993; Oksanen and Visala, 2009; Cabreira et al., 2019; Almadhoun et al., 2019), though identifying and implementing a multiagent coverage algorithm that is tolerant to quadrotor failures to compare with the deep guidance system is saved for future work. This handcrafted zigzag approach is compared to the single-quadrotor deep guidance approach in the outdoor flight experiments in Sec. 5.3.

3. Experimental Facilities

The simulated results are motivated by their corresponding experimental facilities; therefore, a brief overview of the two experimental facilities is presented for context.

3.1. Indoor experiment facility

The indoor experimental facility at ENAC consists of a 10 m × 10 m × 9 m flight volume with a mesh exterior, as shown in Figure 5a. A 16-camera Optitrack system is used to track the motion of the quadrotors with sub-millimeter resolution in real time—a feature that is replaced by considerably less accurate GPS when flying outdoors. The quadrotors are called Explorer 1 and Explorer 2, and are shown in Figure 5b. Including the battery, their mass is 535 g and their maximum thrust is 40 N. A three-cell battery operates at 11.1 V and contains 2,300 mAh, which provides roughly 15 min of flight time.

The indoor quadrotors use the *Paparazzi Autopilot System* (Hattenberger et al., 2014), an open-sourced software package for unmanned aerial systems. *Paparazzi* consists of a ground segment, running on a personal computer, an airborne segment, running on-board the quadrotor, and an XBee communication link between them. The on-board computer is the *Tawaki*; detailed information is presented in Appendix B.

3.2. Outdoor experimental facility

Outdoor runway experiments are tested in a local RC airfield runway located in Muret, France, that is 124 m × 12.5 m in size, as shown in Figure 6a. ENAC has privileged access to this airfield and can do tests in a volume of 500 m radius and up to 150 m height (which can be increased to

(a) Outdoor runway

(b) Outdoor quadrotors and ground station computer

Figure 6. ENAC outdoor facility and flight hardware.

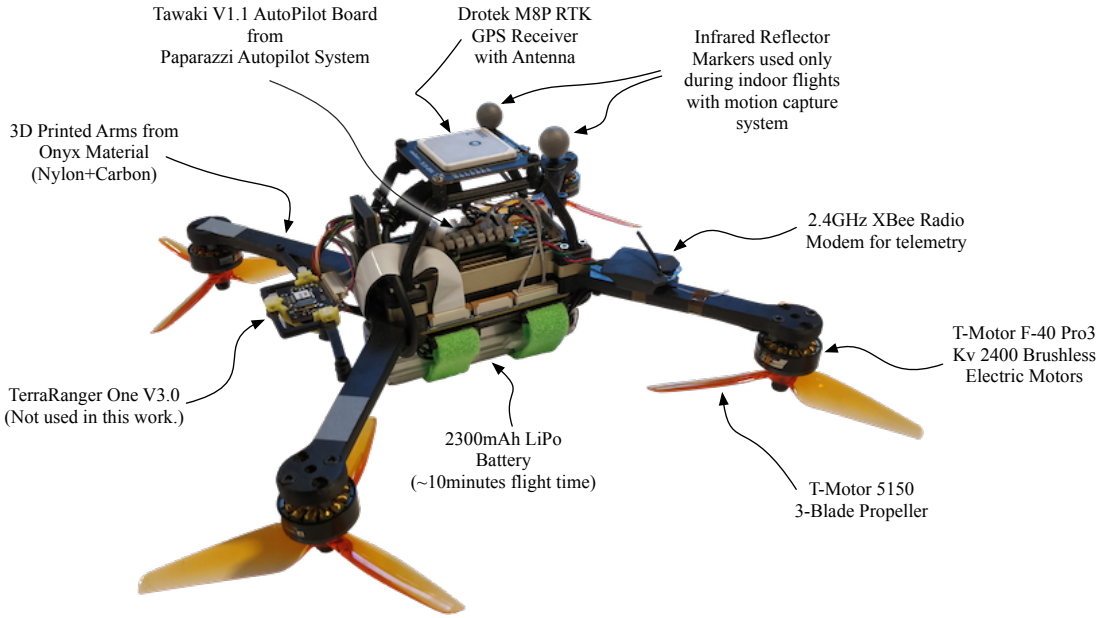


Figure 7. Closeup view of the outdoor quadrotor and its components.

450 m in certain cases). Different quadrotors are designed and manufactured to be used outdoors. They consist of the same avionics and a propulsion system used on the indoor quadrotors but are additionally equipped with a Ublox-M8 GPS receiver which supplies 5 Hz position information with an accuracy of approximately 1.5 m. Due to roughly 150 g of increased mass, the outdoor quadrotors have a reduced flight time of 10 min compared to the indoor quadrotors' 15 min. The quadrotors that are used for the outdoor flights are shown in Figure 6b alongside with the ground control station (an ordinary laptop), safety-pilot transmitters, and an XBee radio modem, which is used for telemetry and down-link communication. Figure 7 shows more quadrotor details, while Figure 8 presents, at a high level, how the signals flow between the various components.

4. Simulation results

To determine whether the real-time closed-loop deep guidance technique is effective at solving the multiquadrotor runway inspection problem, the system is trained in simulation. The simulated

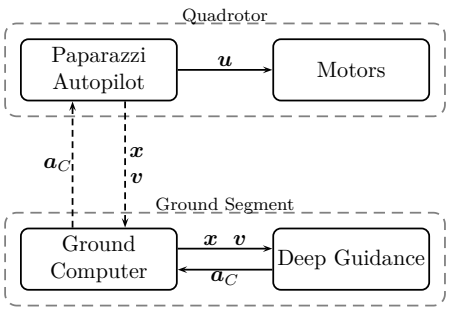


Figure 8. Communication between the various segments in the outdoor experiments. Dashed arrows between the quadrotor and ground computer represent wireless communication using XBee. Indoor experiments obtained $\boldsymbol{x}$ and $\boldsymbol{v}$ from a ground truth system instead of an on-board GPS module.

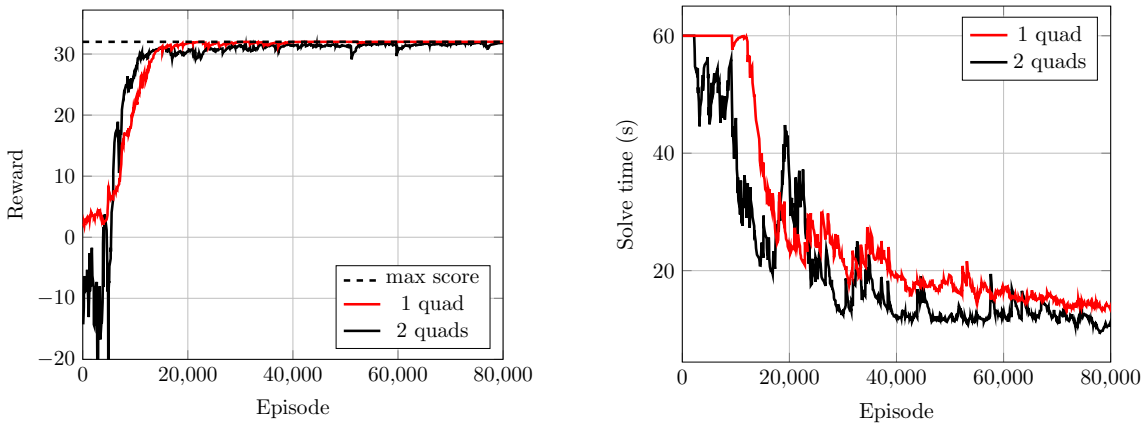
environment is chosen to reflect the available hardware present, discussed in Sec. 3, where the trained policies will ultimately be evaluated in experiment. With the hardware available in mind, the following simulations are designed:

1. An indoor runway inspected by one quadrotor.
2. An indoor runway inspected by two quadrotors.
3. An outdoor runway with one quadrotor inspecting it.
4. An outdoor runway with two quadrotors inspecting it, used to test the fault-tolerant ability of the system to complete the task when one quadrotor fails, as discussed in Sec. 2.2.4.
5. The ability to inspect nonrectangular areas with two quadrotors.
6. An outdoor runway with four and nine times the number of tiles to inspect. Two quadrotors are used.
7. An outdoor runway with a quadrotor swarm (3, 5, 8, and 12 quadrotors) inspecting it. Using 3+ quadrotors is not replicable experimentally but is used to test the limits of the technique discussed here.

4.1. Indoor simulated training results

The indoor runway is chosen to be 4×4 m, which is divided into $S_W = 4$ tiles along its width and $S_L = 8$ tiles along its length, as shown in Figure 1a. The indoor and outdoor runways are divided into the same number of tiles for consistency. The acceleration bounds are set at ± 2 m/s^2, and the maximum velocity is limited to ± 4 m/s. For each episode the quadrotor initial positions are uniformly randomized across the runway to force the policy to learn the behavior from any starting configuration. The agent is allotted a maximum of 300 time steps (60 s) to fully inspect the runway. For the dual-quadrotor task, proximity penalty constants of $c_1 = 1$ and $c_2 = 0.43$ are used such that a penalty of 0.01 per time step is applied when the quadrotors are 2 m apart. Training occurs in the kinematics environment described in Sec. 2.1.2, and the policy is occasionally deployed to the dynamics environment for evaluation. The dynamics environment, discussed in Sec. 2.1.4, uses a mass of 0.5 kg and its controller has $\boldsymbol{K}_I = \text{diag}\{0.5, 0.5\}$, chosen by trial and error, to yield adequate tracking of the guided acceleration signal. All learning plots presented have been exponentially smoothed with a factor of 0.9.

Learning curves are shown in Figure 9a, and solve time curves are shown in Figure 9b. Training results for the single- and dual-quadrotor cases are included on the plots to allow for comparison. The learning curve shows the total rewards received by the agent when its performance is evaluated in a dynamics environment as a function of the number of training episodes completed. The performance quickly increases to its maximum of 32, indicating that runway inspection task has been successfully learned in both in the single- and dual-quadrotor indoor environments. During this initial learning phase, the solve time remains at its maximum, indicating the time limit is reached before the runway



(a) Learning curve **(b)** Solve time

Figure 9. Indoor single-quadrotor runway deep guidance training progress. The learning curve shows the average reward received as a function of episodes and increases as training continues, as expected. The solve time shows the average time needed to solve each episode and decreases as training continues, as expected.

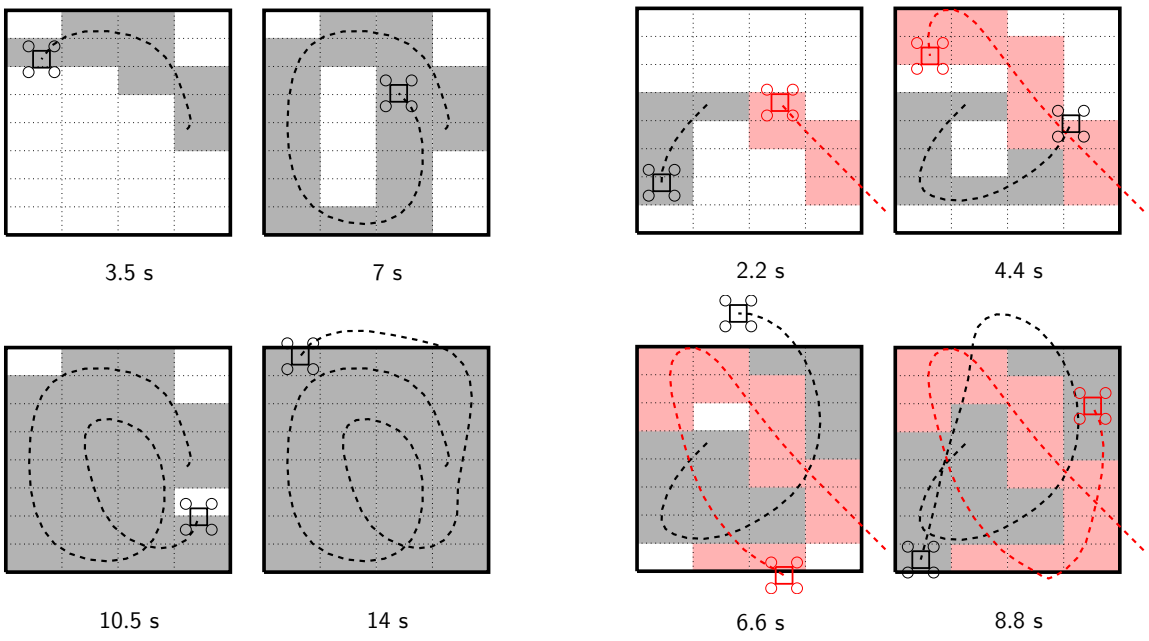
is fully inspected, i.e., the learning has not yet succeeded. Once the ability to fully inspect the runway was learned, further training reduced the amount of time needed to complete the task, as shown in Figure 9b. The discount factor in the reinforcement learning algorithm causes future rewards to be weighted lower than current rewards, which drives the agent to collect rewards as fast as possible and inspect the runway quickly. The average time required to solve once training was complete was 14 s and 10.8 s for the single- and dual-quadrotor scenarios, respectively, indicating that, as expected, two quadrotors can inspect the runway faster than one quadrotor. A single-quadrotor sample trajectory over time is shown in Figure 10a, and a dual-quadrotor sample trajectory over time is shown in Figure 10b. The runway tiles are shaded as they become inspected.

With runway inspection behaviour successfully learned for the single- and dual-quadrotor scenarios indoors, outdoor training simulations are performed next.

4.2. Outdoor single- and dual-quadrotor simulated training results

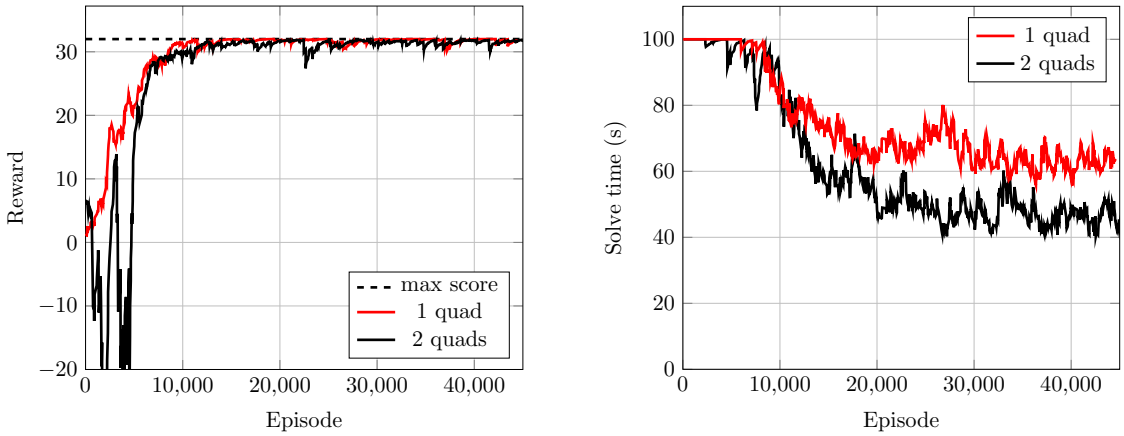
The outdoor runway, presented in Sec. 3.2, is divided into grid tiles similar to those shown in Figure 1b. For these outdoor simulations, the acceleration and velocity bounds are increased to ± 2.5 m/s^2 and ± 5 m/s, respectively. The proximity penalty constants are changed to $c_1 = 1$ and $c_2 = 4.3$ such that a penalty of 0.01 per time step is applied when the quadrotors are 20 m apart to encourage independent inspection. The maximum number of time steps allotted is increased to 500 (100 s). When two quadrotors are used, a 50% chance of one quadrotor failing is included. When a quadrotor fails, its velocity is forced to 0, its position is forced to one corner of the runway, and the data that quadrotor generates is no longer included in the replay buffer for training purposes as discussed in Sec. 2.2.4. Figure 11 shows the learning curve and solve time plot for the outdoor runway inspection with both one and two quadrotors. The outdoor simulated performance is similar to the previous indoor performance. With random initial conditions, the two quadrotors can successfully calculate guidance signals to cooperatively inspect the runway environment without being explicitly programmed. In addition, the runway can reliably be fully inspected when one quadrotor has failed. A single-quadrotor sample trajectory is shown in Figure 12a, a dual-quadrotor sample trajectory without a failure is shown in Figure 12b, and a dual-quadrotor sample trajectory with a failure is shown in Figure 12c.

Two quadrotors are also used to inspect the C- and L-shaped areas discussed in Sec. 2.1.5. Sample trajectories of the two nonrectangular inspection tasks are shown in Figure 13. The quadrotors successfully coordinate their behavior to explore the nonrectangular areas quickly and safely. The



(a) One quadrotor **(b)** Two quadrotors

Figure 10. Indoor simulated runway inspection results. As the quadrotors inspect tiles, they become shaded. The task is considered complete once all tiles become shaded. Quadrotor trajectories are also shown as dashed lines.



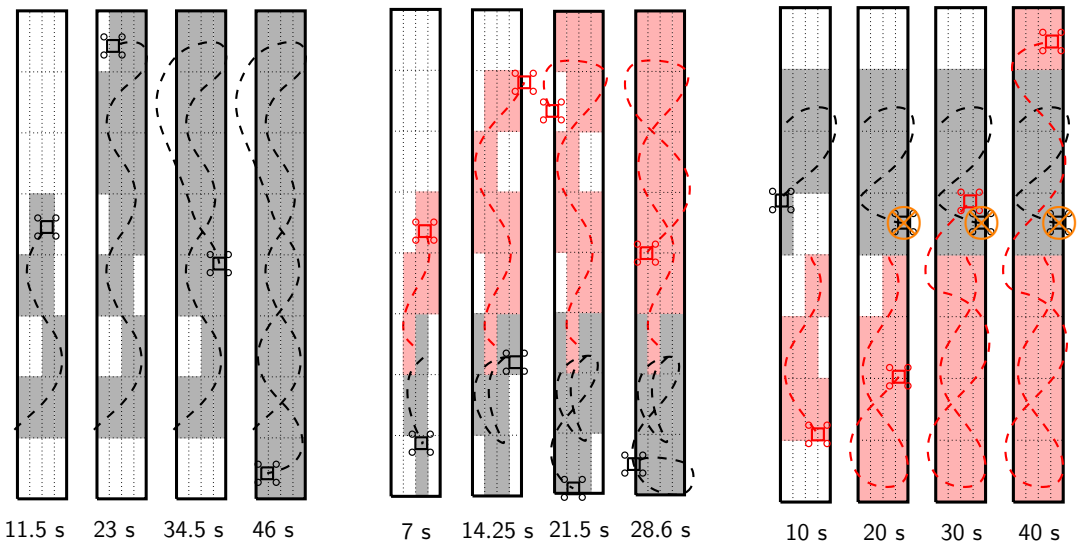
(a) Learning curve **(b)** Solve time

Figure 11. Outdoor single- and dual-quadrotor runway deep guidance training progress. The learning curve shows the average reward received as a function of episodes and increases as training continues, as expected. The solve time shows the average time needed to solve each episode and decreases as training continues, as expected.

following simulation is used to test the limits of the deep guidance algorithm by increasing the grid size and number of quadrotors.

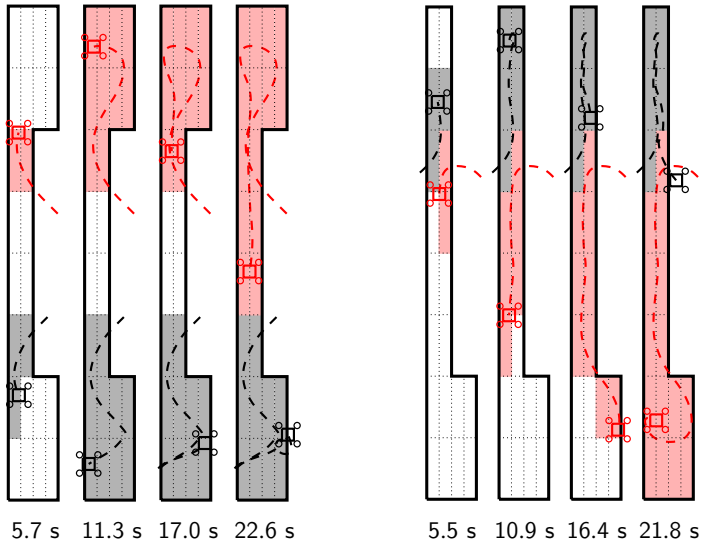
4.3. Grid-size sensitivity

To determine the sensitivity of the deep guidance technique to the resolution of the tiles the runway is divided into, additional simulations are performed with four and nine times the number of runway



(a) One quadrotor **(b)** Two quadrotors **(c)** Two quadrotors with failure

Figure 12. Visualization of sample trajectories from the outdoor simulated training scenarios. Quadrotors are enlarged to show their current position. As the quadrotors inspect tiles, they become shaded. The task is considered complete once all tiles become shaded. Quadrotor trajectories are also shown as dashed lines.

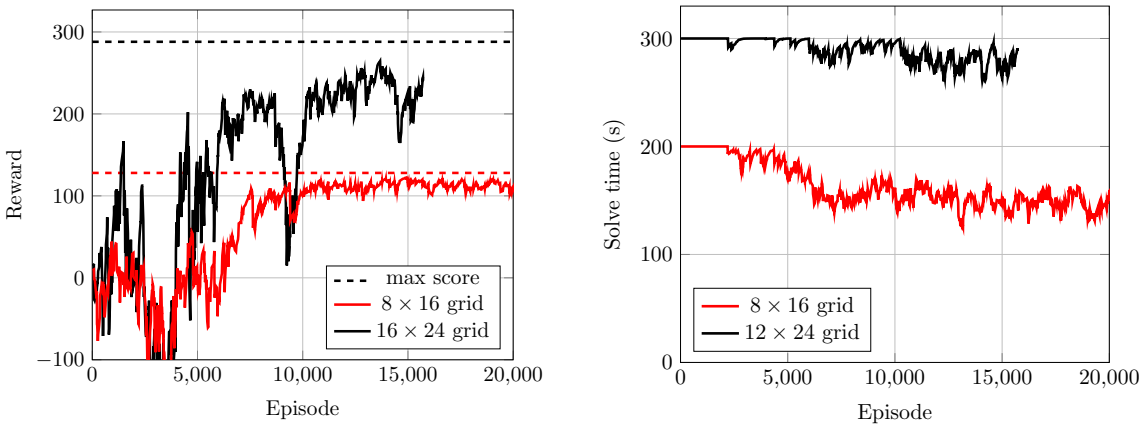


(a) C-shaped area **(b)** L-shaped area

Figure 13. Visualization of sample trajectories from the outdoor simulated nonrectangular area training scenarios. Quadrotors are enlarged to show their current position. As the quadrotors inspect tiles, they become shaded. The task is considered complete once all tiles become shaded. Quadrotor trajectories are also shown as dashed lines.

tiles, i.e., the runway grid is expanded from 4×8 to 8×16 and 12×24. This causes the observation to increase from 40 to 136 or 296. The maximum time allowed to inspect the runway is increased from 100 s to 200 or 300 s, respectively. Learning curves are shown in Figure 14.

Learning succeeds despite the larger runway grid size and the greatly enlarged observation size. During training, the 8×16 grid successfully completes the exploration task on 80% of the attempts



(a) Learning curve

(b) Solve time

Figure 14. Training results with a larger runway grid size. The learning curve shows the average reward received as a function of episodes and increases as training continues, as expected. The solve time shows the average time needed to solve each episode and decreases as training continues, as expected.

and the 12 × 24 grid successfully completes its task on 34% of attempts (though it is possible that more training time would further improve this success rate as the learning curve has not yet plateaued). Both the larger grid sizes are less reliable than with the 4 × 8 runway grid size where the success rate was 97%. Both models are exported for hardware experiments in Sec. 5.3.

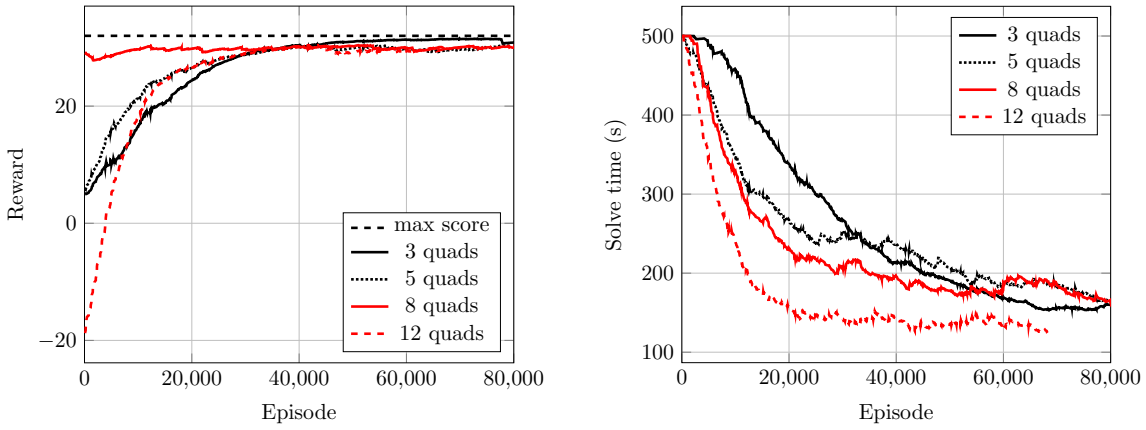
4.4. Quadrotor swarm simulated training results

Although only two quadrotors are available for experiments, the guidance algorithm is trained with additional quadrotors in this section to examine how well the system scales to many quadrotors. The original outdoor runway is used and tested with 3, 5, 8, and 12 quadrotors. Quadrotor failures are not simulated, and the proximity penalty coefficient c_2 in Eq. (9) is reduced to 2.15 for the three-quadrotor scenario and to 1.08 for the remaining scenarios to allow for more quadrotors to inspect the runway without being as penalized for proximity. The learning plots are shown in Figure 15.

The learning curves show that the runway inspection task can be successfully learned with up to 12 quadrotors, despite the observation in Eq. (5) growing significantly in size. The observation has 50 elements when two quadrotors are used and 110 elements when 12 quadrotors are used. Additional quadrotors needed less time to fully inspect the runway than the dual-quadrotor approach. However, the learned behavior with higher numbers of quadrotors is not as creative—all quadrotors fly a similar trajectory left and right on the runway. Since so many quadrotors are present and randomly distributed along the runway, this simple behavior in large numbers is sufficient to inspect the runway. These simulated results represent a scenario where the learning-based approach exploited the reward function. Technically, the left-right behavior seen by all agents is effective at inspecting the entire runway, but it is not the coordinated behavior that was intended. Future work should explore this quadrotor swarm domain further. The single- and dual-quadrotor indoor and outdoor simulations are experimentally validated in the following section.

5. Experimental validation

To explore if the deep guidance technique detailed and simulated above will allow for policies trained entirely in simulation to be transferred to real robots, the experimental facilities presented in Sec. 3 are used. The experimental setup is first presented, followed by the experimental results.



(a) Learning curve **(b)** Solve time

Figure 15. Training results with higher numbers of quadrotors. The exponential smoothing factor is increased to 0.99 for clarity. The learning curve shows the average reward received as a function of episodes and increases as training continues, as expected. The solve time shows the average time needed to solve each episode and decreases as training continues, as expected.

5.1. Experimental setup

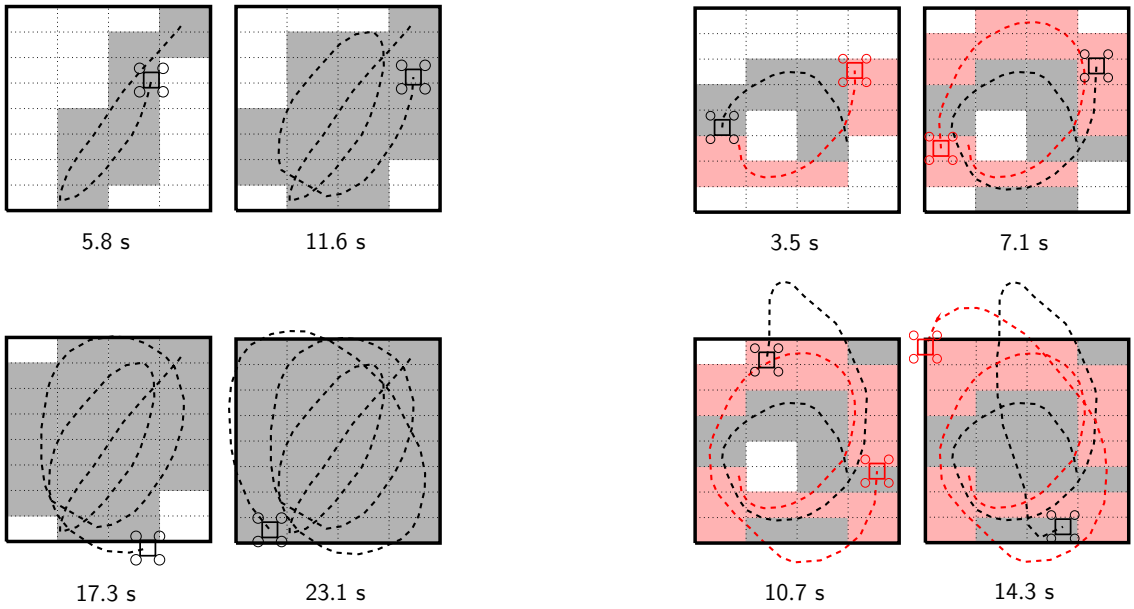
The indoor simulations presented in Sec. 4 were designed to be replicated in the indoor and outdoor experimental facilities presented in Sec. 3. The parameters $\boldsymbol{\theta}$ from the policies trained in simulation are directly exported for use in experiment. The policy input observation in Eq. (5) is assembled using experimental data. Position and velocity information are obtained from the Optitrack ground truth system for indoor flights and from GPS measurements for outdoor flights.

The quadrotors take off and manoeuvre to their randomly assigned holding positions using the autopilot software. Then the quadrotors are switched into deep guidance mode, where they begin listening for real-time deep guidance acceleration signals from the policy network. The on-board controller uses incremental nonlinear dynamic inversion control (Smeur et al., 2016a) to track that acceleration signal. Once the runway is fully inspected, the quadrotors return to their holding positions.

There are many discrepancies between the simulated environments within which the policies were trained and the experimental facilities where their performance is evaluated. The simulated environments did not model the vehicles as quadrotors—they were modelled as double-integrator point masses. Rotor dynamics, air disturbances, actuator limitations, wind, GPS, IMU, ground truth, barometer inaccuracies, and coupling between altitude control and translational control were unmodelled. In addition, the mass and the on-board controllers that track the guided acceleration signals (incremental nonlinear dynamic inversion controller) were different than the ones used during evaluation of the policy performance in simulation (integral controller). Dramatic discrepancies exist between the simulated and experimental environments. Therefore, this is an excellent test of the simulation-to-reality capabilities of the deep guidance technique.

5.2. Indoor experimental results

The initial conditions for each experiment were randomized. The indoor single-quadrotor runway inspection experiments completed their task in 21.1 s on average, with a standard deviation of 3.6 s over nine trials. Nine out of the ten attempts were deemed successful, with one attempt requiring more than the maximum amount of time allotted to finish. A successful single-quadrotor sample trajectory is shown in Figure 16a. The indoor dual-quadrotor runway experiments completed the task in 15.7 s on average, with a standard deviation of 6.9 s over ten trials. All dual-quadrotor experiments



(a) One quadrotor **(b)** Two quadrotors

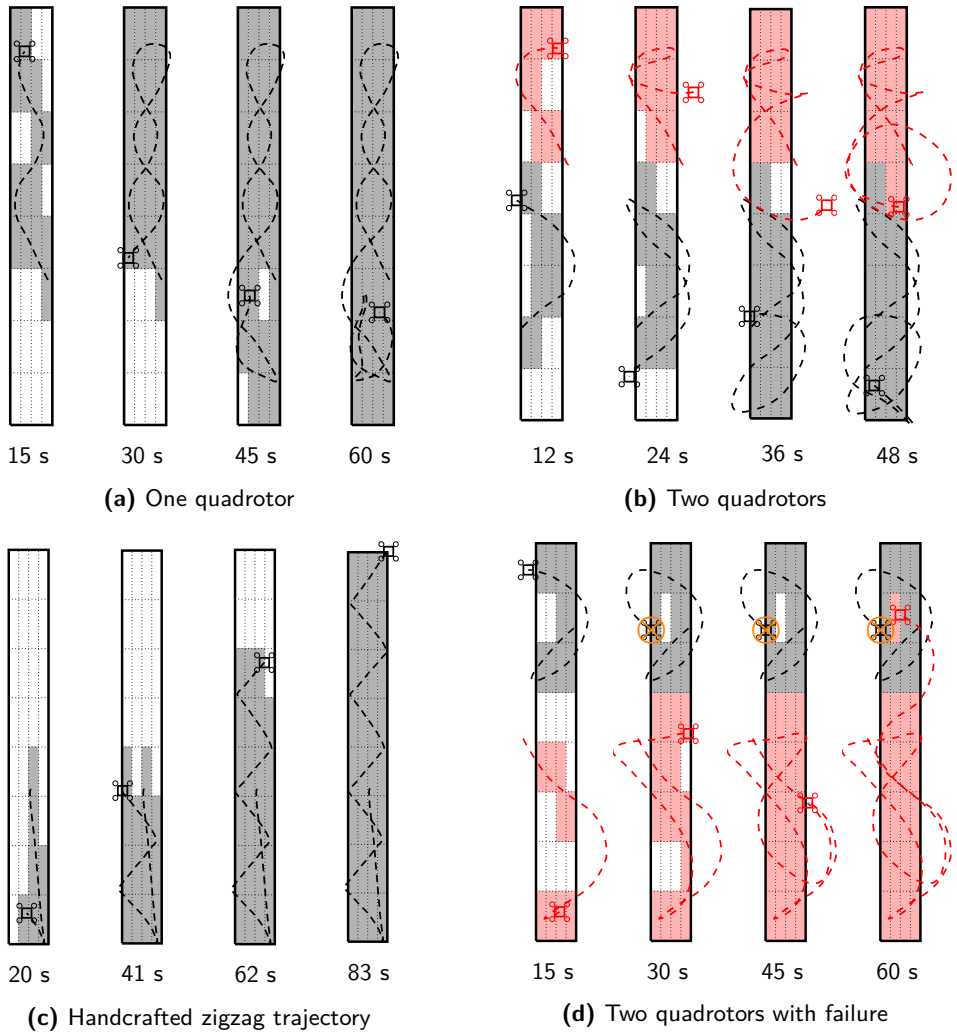
Figure 16. Indoor experimental results. As the quadrotors inspect tiles, they become shaded. The task is considered complete once all tiles become shaded. Quadrotor trajectories are also shown as dashed lines.

were successful; a sample dual-quadrotor trajectory is shown in Figure 16b. The experimental results confirmed that the average time needed to fully inspect the runway is reduced when two quadrotors are used, as expected. The experimental results also confirm that the deep guidance technique is viable for bridging the simulation-to-reality gap and successfully show the application of learning-based guidance techniques to real quadrotors. The indoor experimental campaign overall took roughly 50% more time to complete the runway inspection than in simulation during training, which is attributed to the significant differences between the simulated and experimental environments listed in Sec. 5.1. In addition, communication issues resulting in packet loss, evident by the straight trajectory segments in Figure 16, likely led to some tiles being missed, requiring additional flight time to inspect them.

5.3. Outdoor experimental results

The outdoor single-quadrotor experiment solved the runway inspection task in 76.6 s on average, with a standard deviation of 10.6 s over five trials with randomized initial conditions. Compared to the simulated environment, the experiments took 20% more time to complete the inspection. A representative trajectory is shown in Figure 17a. The zigzag manoeuvre, presented in Sec. 2.2.5, was also tested using randomized initial conditions and is used to compare a handcrafted approach to the learned approach. The zigzag trajectory took 88.8 s on average, with a standard deviation of 5.7 s over five trials. A sample trajectory is shown in Figure 17c.

The single-quadrotor outdoor experimental result shown in Figure 17a resembles the handcrafted zigzag trajectory shown in Figure 17c in that it oscillates left and right as it travels along the runway. Each runway tile only needs to be touched for it to qualify as being inspected—notice how the trajectory barely touches all tiles in the third row from the top of the runway. In comparison, the zigzag approach was inflexible to changes in initial conditions—it simply followed the prescribed set of waypoints. A waypoint is considered "reached" when the quadrotor is within 2.5 m of it. Still, the quadrotor loses significant speed at each waypoint, through making sharp turns, which causes the zigzag approach to complete the inspection slower than the deep guidance approach. The



(a) One quadrotor

(b) Two quadrotors

(c) Handcrafted zigzag trajectory

(d) Two quadrotors with failure

Figure 17. Outdoor experimental results. As the quadrotors inspect tiles, they become shaded. The task is considered complete once all tiles become shaded. Quadrotor trajectories are also shown as dashed lines.

single-quadrotor deep guidance approach effectively learned a modified zigzag pattern that is able to maintain its speed throughout the flight while automatically adapting its flight path to every initial condition encountered.

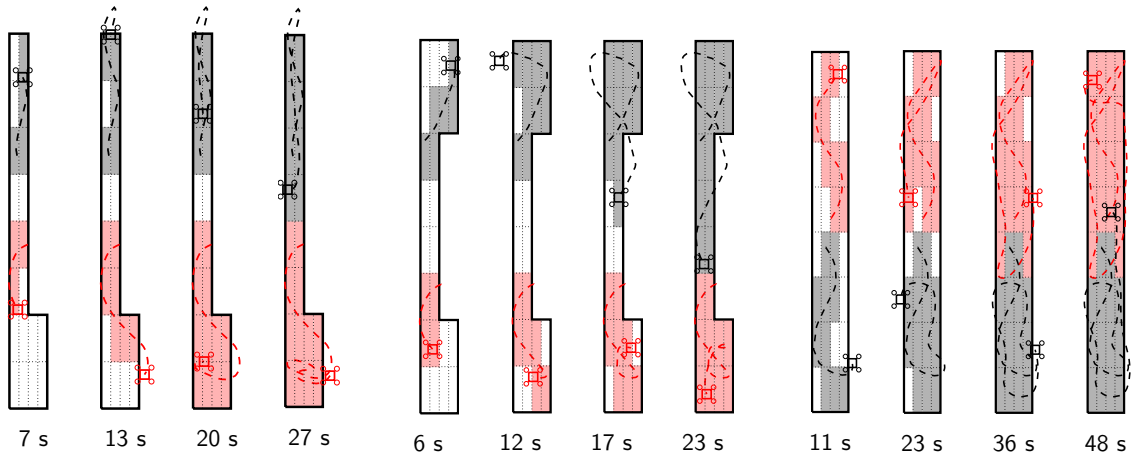
The dual-quadrotor outdoor experiments were performed with four different initial conditions. They fully inspected the runway in 56.5 s on average, with a standard deviation of 11.8 s, taking 25% more time than in simulation. The increase in solve time during the outdoor experiments, compared to the training simulations, was lower than the indoor experiments' 50% increase. Both the indoor and outdoor experiments used an XBee communications system, but the outdoor flights suffered from less packet loss than the indoor flights, leading to one less barrier between simulation and reality for the deep guidance system to overcome, and is the likely source of the improvement. A sample trajectory is shown in Figure 17b. Following this, four more trials were flown with a simulated failure of one quadrotor occurring at 20 s into the inspection. That is, one quadrotor fails while the other one must continue to inspect the runway. In all cases, the remaining quadrotor finished the inspection task on its own, without any human intervention, demonstrating the flexibility of the deep guidance approach. The average solve time with a failure rose to 70.0 s on average, with a standard deviation of 13.4 s. A sample trajectory is shown in Figure 17d.

As illustrated in Figure 17b, the red quadrotor opted to cover the top portion of the runway while the black quadrotor chose to inspect the bottom portion. This is despite that the red quadrotor started near the center of the runway and therefore could have inspected in either direction. The cooperative nature of the system is demonstrated—the quadrotors choose their behavior based on the anticipated behavior of the other, the state of each quadrotor, and the state of the runway. In the case presented in Figure 17d, notice how the initial conditions were swapped. Here, the black quadrotor chose to inspect the top portion of the runway while the red quadrotor inspected the bottom portion. The same trained policy was used in both experiments that generated Figs. 17d and 17b, demonstrating the wide range of behaviors that are generated from a single policy. In addition, one quadrotor suffered a failure in Figure 17d. Upon sensing the failure, the still-operational red quadrotor completed the inspection of its portion of the runway before inspecting the final tile on the upper portion of the runway. This clearly demonstrates the flexibility of the dual-quadrotor deep guidance system to varying initial conditions and quadrotor failures.

To verify that the fault tolerance must be seen during training in order to have the capability realized in experiment, additional dual-quadrotor experiments were performed using a deep guidance model that did not encounter quadrotor failures during training. When failures were then experienced during outdoor runway experiments, the surviving quadrotor did not successfully complete the runway inspection.

Experimental results are obtained for an L-shaped runway, a C-shaped runway, and when penalties are assigned for leaving the runway, shown in Figure 18. The L- and C-shaped runways are successfully inspected in a similar fashion to the simulated results presented in Sec. 13. With additional penalties assigned for leaving the runway proper, in Figure 18c the quadrotors complete their inspection while leaving the runway less frequently. Experimental results with a 4×- and 9×-grid size (discussed in Sec. 4.3) were successfully obtained, though their trajectories are not included due to their lack of clarity.

Statistics on all the simulated and experimental runs performed are reported in Table 1. The average solve time in experiment is always longer than that of the simulation. A variety of factors that impact the quadrotors in experiment that are not encountered in simulation, described in Sec. 5.1, are the attributed cause. The average speed in experiment is typically larger than in simulation due to problems with accurately capping the maximum velocity of the quadrotors in experiment. The average distances again are typically longer in experiment than in simulation due to the above-mentioned reasons (in addition to intermittent communication issues on trials marked with an * that inflate the averages). Regardless, the trends between simulation and experiment are comparable.



(a) L-shaped runway **(b)** C-shaped runway **(c)** Penalties for leaving runway

Figure 18. Additional outdoor experimental results. As the quadrotors inspect tiles, they become shaded. The task is considered complete once all tiles become shaded. Quadrotor trajectories are also shown as dashed lines.

Table 1. Overall statistics of the flight tests. Definitions: 1Q: one quadrotor; 2QN: two quadrotors without failures; 2Q: two quadrotors (failure tolerant); 2QF: two quadrotors (failure intolerant); 2Q4x & 2Q9x: two quadrotors with 4x and 9x the number of grid tiles, respectively; 2QC & 2QL: two quadrotors on the C- and L-shaped runways, respectively; 2QEP: two quadrotors with extra penalties for leaving the runway. Values marked with a * have been inflated due to intermittent communication issues, and 2QF did not complete any experiments and therefore some values are not applicable.

	Indoor Flights		Outdoor Flights								
	1Q	2QN	1Q	2QN	2Q	2QF	2Q4x	2Q9x	2QC	2QL	2QEP
# Experiments	10	10	5	4	4	6	5	1	5	5	6
Success (%)	90	100	100	100	100	0	100	100	100	100	100
Avg Time Sim (s)	14	11	63.6	38.5	44.7	∞	157	281	27.6	25.6	38.8
Avg Time Exp (s)	21.2	15.7	76.6	56.5	70.0	∞	248	528	51.9	42.0	47.7
Avg Vel Sim (m/s)	1.68	1.58	5.06	4.88	4.26	4.88	5.08	5.21	4.19	4.21	4.12
Avg Vel Exp (m/s)	1.13	0.69	2.95	5.1	5.49	5.91	5.59	5.94	5.25	4.96	4.69
Avg Dist Sim (m)	24.6	38.7	310	367	380	–	1507	2987	240	232	331
Avg Dist Exp (m)	24.0	21.7	225	574	481	–	2777*	6271*	568*	426*	449*

The complexity of the learned behaviors has emerged from a simple reward function: +1 for inspecting a new tile and penalties for quadrotor proximity. Though it was not attempted, it would likely require significant engineering effort to handcraft a dual-quadrotor guidance algorithm that can similarly (a) handle any initial condition efficiently and (b) gracefully adjust when one quadrotor fails. These results demonstrate the power of learning-based approaches to solve tasks specified at a high level—"inspect the runway" in this paper—and the deep guidance technique, which permits only guidance to be learned, allows for these behaviors to be realized not only in simulation but also in a representative, outdoor, GPS-driven facility, providing a possible solution to the simulation-to-reality gap.

Learning-based methods have their limitations: learning can be difficult to achieve, performance cannot be guaranteed, and significant computing power (during the training phase) is required. A video summarizing the simulations and experiments can be found at https://youtu.be/Pu5rWnLgyZs.

6. Conclusions

This paper used deep reinforcement learning to act as closed-loop real-time guidance for quadrotors to autonomously and cooperatively learn to perform a runway inspection. A simple reward scheme was used that encouraged cooperative runway inspection guidance to be learned without any human intervention or design, significantly reducing the engineering effort required to complete the task. The multiquadrotor runway inspection system was trained entirely in simulation and deployed to two real-world experimental facilities at École Nationale de l'Aviation Civile, where both indoor and outdoor GPS-driven experiments were performed. The indoor and outdoor experiments were successful, though they required 50% and 25% more time, respectively, to complete the inspection than during training in simulation. There were significant discrepancies between the simulated environment within which the deep guidance policies were trained and the experimental environment where they were evaluated. This highlights the simulation-to-reality capability of the deep guidance approach: using deep reinforcement learning for guidance is a potential avenue for allowing policies trained exclusively in simulation to be executed on real hardware.

The deep guidance approach was shown to complete the task faster than a handcrafted zigzag approach, create successful trajectories regardless of its initial conditions, and, in the dual-quadrotor scenario, be tolerant of the failure of one quadrotor. The dual-quadrotor model learned to divide the runway into portions with each quadrotor assuming responsibility for the nearest area, which varied from experiment to experiment since the initial conditions were randomized. The deep guidance system was shown to be trivial to retrain for use on nonrectangular inspection areas by removing

rewards from irrelevant tiles. Both C- and L-shaped runways were used to show that this approach can be used as a more general distributed inspection system—not only for rectangular runways. Lastly, learning still succeeded, though with more difficulty, despite significant increases in runway grid size and number of quadrotors used.

6.1. Lessons learned and future work

While reinforcement learning is able to produce impressive multiagent results with little engineering effort (i.e., only specifying the reward function), effort is required to tune all the relevant learning parameters, α, β, γ, ϵ, σ, M, and N, among others. An incorrectly chosen learning parameter may cause the learning algorithm to fail. Careful attention to the reward function is also important. The learning algorithm has no sense of the desired behavior and is only driven to maximize the rewards received; the desired behavior is simply a by-product of this reward-seeking behavior. Therefore, the reward function must be designed such that it cannot be exploited, as was the case in the quadrotor swarm scenario. The quadrotor swarm scenario learned an exploration strategy that blindly guided all agents left and right over the runway, using their large numbers to explore the entire runway instead of truly learning a coordinated exploration strategy. Lastly, during preliminary experiments the acceleration signals were not being properly tracked, which put the quadrotors into states not often encountered during training (far outside the runway). Since the learning algorithm had never encountered quadrotors very far from the runway during training, it did not learn a suitable behavior to recover the flight. Although learning algorithms can generalize the examples seen during training to new unseen scenarios, those new scenarios must still be within the range of scenarios encountered during training. It is therefore equally important for the controller used in experiment, which may be different than the one used in simulation, to track the guided accelerations as closely as possible.

During the experiments, the communication link was occasionally lost between the ground station computer and the quadrotors. This caused the quadrotor to continue with its last-commanded acceleration, often carrying the quadrotor well outside the runway. However, when the signal was regained, the quadrotor recovered and completed its mission. Flight speed was another difficulty to manage during the outdoor experiments. The maximum velocity limits were not always respected, which yielded states not seen during training (velocities higher than their maximum). Further improvements could be made with link loss and flight speed management.

Future work should capture images at each tile and merge them into a composite runway image for processing, modify the observation such that it does not grow with increased quadrotors, investigate performance guarantees of learning-based approaches, compare the performance to currently available coverage algorithms, or develop an algorithm that can inspect an arbitrary shape without retraining.

Acknowledgments

This research was financially supported in part by the Natural Sciences and Engineering Research Council of Canada under the Postgraduate Scholarship-Doctoral PGSD3-503919-2017 and Ontario Graduate Scholarship awards. The authors would like to thank Xavier Paris for his support during both the indoor and outdoor flight experiments. This research was enabled in part by support provided by Research Computing Services[2] at Carleton University, with specific thanks to Ryan Taylor for his help.

ORCID

Kirk Hovell https://orcid.org/0000-0003-3519-2922
Steve Ulrich https://orcid.org/0000-0001-7465-8786
Murat Bronz https://orcid.org/0000-0002-1098-5240

[2] https://carleton.ca/rcs

References

Absolon, S., Hůlek, D., Trešlová, H., and Skálová, M. (2015). Runway Inspection by RPAS. *Magazine of Aviation Development*, 3(16).

Almadhoun, R., Taha, T., Seneviratne, L., and Zweiri, Y. (2019). A Survey on Multi-robot Coverage Path Planning for Model Reconstruction and Mapping. *SN Applied Sciences*, 1(847).

Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., TB, D., Muldal, A., Heess, N., and Lillicrap, T. (2018). Distributed Distributional Deterministic Policy Gradients. In *International Conference on Learning Representations*, Vancouver, Canada.

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A Distributional Perspective on Reinforcement Learning. In *International Conference on Machine Learning*, pages 449–458, Sydney, Australia. PMLR.

Buşoniu, L., Babuška, R., and De Schutter, B. (2008). A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 38(2):156–172.

Cabreira, T. M., Ferreira, P. R., Franco, C. D., and Buttazzo, G. C. (2019). Grid-based Coverage Path Planning with Minimum Energy Over Irregular-shaped Areas with UAVs. In *International Conference on Unmanned Aircraft Systems*, pages 758–767, Piscataway, NJ. IEEE.

Camci, E. and Kayacan, E. (2019). End-to-end Motion Planning of Quadrotors Using Deep Reinforcement Learning. *Computing Research Repository*.

Chu, X. and Ye, H. (2017). Parameter Sharing Deep Deterministic Policy Gradient for Cooperative Multiagent Reinforcement Learning. *Computing Research Repository*.

Cutler, M. and How, J. P. (2016). Autonomous Drifting Using Simulation-aided Reinforcement Learning. In *IEEE International Conference on Robotics and Automation*, pages 5442–5448, Piscataway, NJ. IEEE Publications.

Das, A., Subbarao, K., and Lewis, F. (2009). Dynamic Inversion with Zero-dynamics Stabilisation for Quadrotor Control. *IET Control Theory & Applications*, 3(3):303–314.

Greatwood, C. and Richards, A. G. (2019). Reinforcement Learning and Model Predictive Control for Robust Embedded Quadrotor Guidance and Control. *Autonomous Robots*, 43:1681–1693.

Gupta, J. K., Egorov, M., and Kochenderfer, M. (2017). Cooperative Multi-agent Control Using Deep Reinforcement Learning. In *Autonomous Agents and Multi-Agent Systems*, volume 10642, pages 66–83. Springer.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. (2018). Soft Actor-Critic Algorithms and Applications. *Computing Research Repository*.

Harris, A., Teil, T., and Schaub, H. (2019). Spacecraft Decision-Making Autonomy using Deep Reinforcement Learning. In *AAS/AIAA Space Flight Mechanics Meeting*, Ka'anapali, HI. AAS Paper 19-447.

Hattenberger, G., Bronz, M., and Gorraz, M. (2014). Using the Paparazzi UAV System for Scientific Research. In *International Micro Air Vehicles Conference and Flight Competition*, pages 247–252, Delft, Netherlands.

Hochreiter, S. (1998). The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems*, 6(2):107–116.

Hoffmann, G. M., Waslander, S. L., and Tomlin, C. J. (2008). Quadrotor Helicopter Trajectory Tracking Control. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, Honolulu, HI. AIAA Paper 2008-7410.

Honig, W., Preiss, J. A., Kumar, T. K., Sukhatme, G. S., and Ayanian, N. (2018). Trajectory Planning for Quadrotor Swarms. *IEEE Transactions on Robotics*, 34(4):856–869.

Hovell, K. and Ulrich, S. (2021). Deep Reinforcement Learning for Spacecraft Proximity Operations Guidance. *Journal of Spacecraft and Rockets*, 58(2):254–264.

Hovell, K., Ulrich, S., and Bronz, M. (2021). Acceleration-based Quadrotor Guidance Under Time Delays Using Deep Reinforcement Learning. In *AIAA Guidance, Navigation and Control Conference*, Nashville, TN. AIAA paper 2021-1751.

Hwangbo, J., Sa, I., Siegwart, R., and Hutter, M. (2017). Control of a Quadrotor with Reinforcement Learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103.

James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., and Bousmalis, K. (2019). Sim-to-real via Sim-to-sim: Data-efficient Robotic Grasping Via Randomized-

to-canonical Adaptation Networks. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 12619–12629, Piscataway, NJ.

Julian, K. D. and Kochenderfer, M. J. (2019). Distributed Wildfire Surveillance with Autonomous Aircraft Using Deep Reinforcement Learning. *Journal of Guidance, Control, and Dynamics*, 42(8):1768–1778.

Junell, J. L., Van Kampen, E.-J., de Visser, C. C., and Chu, Q. (2015). Reinforcement Learning Applied to a Quadrotor Guidance Law in Autonomous Flight. In *AIAA Guidance, Navigation, and Control Conference*, Kissimmee, FL. AIAA Paper 2015-1990.

Katsikopoulos, K. V. and Engelbrecht, S. E. (2003). Markov Decision Processes with Delays and Asynchronous Cost Collection. *IEEE Transactions on Automatic Control*, 48(4):568–574.

Kendoul, F. (2012). Survey of Advances in Guidance, Navigation, and Control of Unmanned Rotorcraft Systems. *Journal of Field Robotics*, 29(2):315–378.

Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, San Diego, CA.

Koch, W., Mancuso, R., West, R., and Bestavros, A. (2019). Reinforcement Learning for UAV Attitude Control. *ACM Transactions on Cyber-Physical Systems*, 3(2):1–21.

Lee, J., Hwangbo, J., and Hutter, M. (2019). Robust Recovery Controller for a Quadrupedal Robot using Deep Reinforcement Learning. *Computing Research Repository*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous Control with Deep Reinforcement Learning. In *International Conference on Learning Representations*, San Juan, Puerto Rico.

Loquercio, A., Kaufmann, E., Ranftl, R., Dosovitskiy, A., Koltun, V., and Scaramuzza, D. (2020). Deep Drone Racing: From Simulation to Reality with Domain Randomization. *IEEE Transactions on Robotics*, 36(1):1–14.

Madani, T. and Benallegue, A. (2006). Backstepping Control for a Quadrotor Helicopter. In *IEEE International Conference on Intelligent Robots and Systems*, pages 3255–3260, Piscataway, NJ.

Mellinger, D. and Kumar, V. (2011). Minimum Snap Trajectory Generation and Control for Quadrotors. In *IEEE International Conference on Robotics and Automation*, pages 2520–2525, Piscataway, NJ.

Mellinger, D., Michael, N., and Kumar, V. (2012). Trajectory Generation and Control for Precise Aggressive Maneuvers with Quadrotors. *International Journal of Robotics Research*, 31(5):664–674.

Mnih, V., Badia, A., Mirza, M., Graves, A., and Lillicrap, T. (2016). Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on Machine Learning*, pages 1928–1937, New York, NY. PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-Level Control Through Deep Reinforcement Learning. *Nature*, 518(7540):529–533.

Oksanen, T. and Visala, A. (2009). Coverage Path Planning Algorithms for Agricultural Field Machines. *Journal of Field Robotics*, 26(8):651–668.

Oliphant, T. E. (2007). Python for Scientific Computing. *Computing in Science & Engineering*, 9(3):10–20.

OpenAI (2018). Learning Dexterous In-Hand Manipulation. *Computing Research Repository*.

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. (2019). Solving Rubik's Cube with a Robot Hand. *Computing Research Repository*.

OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W. (2020). Learning Dexterous in-hand Manipulation. *International Journal of Robotics Research*, 39(1):3–20.

Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. In *IEEE International Conference on Robotics and Automation*, pages 3803–3810, Piscataway, NJ.

Sadeghi, F. and Levine, S. (2017). CAD2RL: Real Single-image Flight Without a Single Real Image. In *Robotics: Science and Systems*, Cambridge, MA.

Sappington, R. N., Acosta, G. A., Hassanalian, M., Lee, K., and Morelli, R. (2019). Drone Stations in Airports for Runway and Airplane Inspection Using Image Processing Techniques. In *AIAA Aviation Forum*, Dallas, TX. AIAA Paper 2019-3316.

Shah, K., Ballard, G., Schmidt, A., and Schwager, M. (2020). Multidrone Aerial Surveys of Penguin Colonies in Antarctica. *Science Robotics*, 5(47).

Siddiquee, M., Junell, J., and van Kampen, E. J. (2019). Flight Test of Quadcopter Guidance with Vision-based Reinforcement Learning. In *AIAA Intelligent Systems Conference*, San Diego, CA. AIAA Paper 2019-0142.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2017a). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *Computing Research Repository*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T., and Hassabis, D. (2017b). Mastering the Game of Go Without Human Knowledge. *Nature*, 550(7676):354–359.

Smeur, E. J., Chu, Q., and De Croon, G. C. (2016a). Adaptive Incremental Nonlinear Dynamic Inversion for Attitude Control of Micro Air Vehicles. *Journal of Guidance, Control, and Dynamics*, 39(3):450–461.

Smeur, E. J., De Croon, G. C., and Chu, Q. (2016b). Gust Disturbance Alleviation with Incremental Nonlinear Dynamic Inversion. In *IEEE International Conference on Intelligent Robots and Systems*, pages 5626–5631, Piscataway, NJ.

Smeur, E. J., de Croon, G. C., and Chu, Q. (2018). Cascaded Incremental Nonlinear Dynamic Inversion for MAV Disturbance Rejection. *Control Engineering Practice*, 73:79–90.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge: MIT Press, 2nd edition. pg. 148.

Tan, M. (1993). Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. In *International Conference on Machine Learning*, pages 330–337, Amherst, MA.

Terry, J. K., Grammel, N., Hari, A., and Santos, L. (2020). Parameter Sharing is Surprisingly Useful for Multi-Agent Deep Reinforcement Learning. *Computing Research Repository*.

Van Baar, J., Sullivan, A., Cordorel, R., Jha, D., Romeres, D., and Nikovski, D. (2019). Sim-to-real Transfer Learning Using Robustified Controllers in Robotic Tasks Involving Complex Dynamics. In *IEEE International Conference on Robotics and Automation*, pages 6001–6007, Piscataway, NJ.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. (2019). Grandmaster Level in StarCraft II Using Multi-agent Reinforcement Learning. *Nature*, 575(7782):350–354.

Wang, Y., Sun, J., He, H., and Sun, C. (2020). Deterministic Policy Gradient With Integral Compensator for Robust Quadrotor Control. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(10):3713–3725.

Xu, R. and Ozguner, U. (2006). Sliding Mode Control of a Quadrotor Helicopter. In *IEEE Conference on Decision and Control*, pages 4957–4962, Piscataway, NJ. IEEE.

Yang, Y., Caluwaerts, K., Iscen, A., Zhang, T., Tan, J., and Sindhwani, V. (2019). Data Efficient Reinforcement Learning for Legged Robots. In *Conference on Robotic Learning*, Osaka, Japan.

Zelinsky, A., Jarvis, R., Byrne, J., and Yuta, S. (1993). Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot. In *International Conference on Advanced Robotics*, pages 533–538, Tsukuba, Japan.

How to cite this article: Hovell, K., Ulrich, S., & Bronz, M. (2022). Learned multiagent real-time guidance with applications to quadrotor runway inspection. *Field Robotics, 2*, 1105–1133.

Appendix A. Deep reinforcement learning and D4PG

This Appendix discusses deep reinforcement learning (DRL) and the D4PG algorithm used in this research.

A.1. Deep reinforcement learning

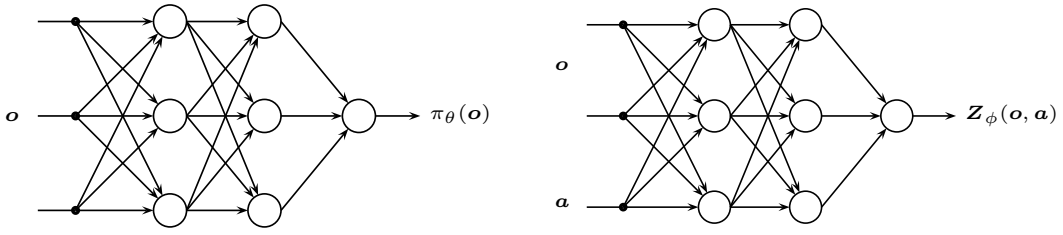
Deep reinforcement learning attempts to discover a policy, π_θ, approximated using a neural network that has trainable parameters, θ, in order to select an appropriate action, $\boldsymbol{a} \in \mathcal{A}$, given the current observation, $\boldsymbol{o} \in \mathcal{O}$, that maximizes the scalar rewards received, r, over time. If the observation is the true state, $\boldsymbol{x} \in \mathcal{X}$, the system is called a Markov decision process. Otherwise, the observation contains information from which the state must be inferred (e.g., an image), in which case the system is called a partially observable Markov decision process. In either case, the action is obtained from the policy, at time step t, through

$$\boldsymbol{a}_t = \pi_\theta(\boldsymbol{o}_t) \tag{1}$$

Of the many available reinforcement learning algorithms, the distributed distributional deep deterministic policy gradient (D4PG) algorithm (Barth-Maron et al., 2018) is used in this work to train the policy. The D4PG algorithm was selected because it allows for continuous state and action spaces, training can be distributed across many CPUs, and its output is deterministic. The following subsection describes the algorithm.

A.2. D4PG algorithm

The D4PG (Barth-Maron et al., 2018) algorithm has an actor-critic architecture, which means that two separate neural networks are used: an actor and a critic. The actor, also known as the policy, with trainable parameters θ, is the component that actually executes within the environment: it calculates which actions should be taken given an observation. The critic, also known as the value network, is a separate neural network, with trainable parameters ϕ, that predicts the performance of the policy. Having an accurate prediction of the policy's performance is a necessity for improving the policy. The value network receives the same observation as the policy, but it also receives the action that the policy took and the resulting rewards that were received from the environment. Using many examples of these data, it can predict how many rewards will be received by taking a certain action from a given observation. Instead of the critic outputting a single prediction of the expected rewards, however, it outputs a probability distribution (the "distributional" aspect of the D4PG name) of the expected rewards from this observation-action input pair, $\boldsymbol{Z}_\phi(\boldsymbol{o}, \boldsymbol{a})$. Figure 19 shows the policy and value neural networks.



(a) Policy neural network **(b)** Value neural network

Figure 19. Policy and value neural networks used in the D4PG algorithm (not to scale).

The total expected reward from a given observation is the expectation over the critic probability distribution. Since the critic uses the action from the policy as an input, the policy is substituted for the action in $\boldsymbol{Z}_\phi(\boldsymbol{o}, \boldsymbol{a})$, giving

$$J(\theta) = \mathbb{E}\{\boldsymbol{Z}_\phi(\boldsymbol{o}, \pi_\theta(\boldsymbol{o})\} \tag{2}$$

where $J(\theta)$ is the expected rewards from the given observation as a function of the policy weights θ and where $\mathbb{E}$ denotes the expectation. Reinforcement learning systematically adjusts θ to discover a policy π_θ that maximizes the expected rewards received, $J(\theta)$. In doing so, the policy indirectly learns a behavior suited to solve the task specified by the reward function.

Data are needed to train the neural networks in order to discover a policy that solves the task. An "on-policy" algorithm requires the data be collected from the most up-to-date version of the policy *only*. An "off-policy" algorithm can use data collected from current and older versions of the policy, making off-policy algorithms more sample efficient. D4PG is an off-policy algorithm. To generate the data needed to train the networks, a representative simulation for the task is developed. At each time step, the policy network is used to choose an action to take as a function of the observation using Eq. (1). The chosen action is executed on the environment, which applies the action while stepping the environment forward one time step. The environment then returns the next observation and a reward. A number of parallel simulations, K, are run simultaneously to generate large amounts of observation, action, next observation, and reward data. The generated data are placed in a replay buffer that holds the most recent R data. Since the D4PG algorithm is off policy, the learning algorithm can randomly sample batches of M data points from the replay buffer to train the policy and value networks using backpropagation and gradient descent. Over time, the performance of the policy, as measured by the amount of rewards it receives over one simulation (also called an episode), is expected to, on average, increase.

A.2.1. Value network training

The value network is trained using stochastic gradient descent to minimize the cross-entropy loss function given by

$$L(\phi) = \mathbb{E}\{-\boldsymbol{Y} \log(\boldsymbol{Z}_\phi(\boldsymbol{o}, \boldsymbol{a}))\} \tag{3}$$

where $\boldsymbol{Y}$ is the target value distribution (Bellemare et al., 2017), which is a better estimate of the true value distribution because it is calculated from data, via

$$\boldsymbol{Y}_t = \sum_{n=0}^{N-1} \gamma^n r_{t+n} + \gamma^N \boldsymbol{Z}_{\phi'}(\boldsymbol{o}_{t+N}, \pi_{\theta'}(\boldsymbol{o}_{t+N})) \tag{4}$$

where r_{t+n} is the reward received at time step $t+n$, γ is the discount factor for future rewards (future rewards are weighted lower than current rewards) and $\boldsymbol{Z}_{\phi'}(\boldsymbol{o}_{t+N}, \pi_{\theta'}(\boldsymbol{o}_{t+N}))$ is the value distribution itself evaluated N time steps into the future. After N time steps of data are used, the remainder of the episode is approximated using the value distribution from $\boldsymbol{o}_{t+N}$ (Mnih et al., 2016). Essentially, the "best guess" at the correct value distribution, which Eq. (4) predicts, is N time steps of evidence (the first term) plus the value network's own prediction for what the future holds (the second term). The parameters θ' and ϕ' are smoothed versions of the true policy and value network weights, θ and ϕ, calculated through

$$\theta' = (1-\epsilon)\theta' + \epsilon\theta \tag{5}$$

$$\phi' = (1-\epsilon)\phi' + \epsilon\phi \tag{6}$$

with $\epsilon \ll 1$. Using exponentially smoothed copies of the policy and value networks when calculating the value distribution estimate in Eq. (4) has a stabilizing effect on the learning (Mnih et al., 2015). In order to train the value network in one iteration, Eq. (4) is used to calculate an updated estimate for the true value network distribution. The loss function is then minimized using Eq. (3) through adjusting ϕ using learning rate β. This causes the value network predictions to slowly approach the value distribution dictated by the simulated data. The value network parameters are then smoothed, using Eqs. (5) and (6), and are again used in Eq. (4)—a recursive process Sutton and Barto described as "*we learn a guess from a guess*" (Sutton and Barto, 1998).

A.2.2. Policy network training

Following each value network training iteration, the policy network is trained in one iteration. Compared to training the critic, which directly uses simulated data and stochastic gradient descent, the policy's training is guided by the critic. The goal is to adjust the policy parameters θ to increase the expected performance, $J(\theta)$. Since neural networks are differentiable, the chain rule can be used to compute

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{\partial J(\theta)}{\partial \boldsymbol{a}} \frac{\partial \boldsymbol{a}}{\partial \theta} \tag{7}$$

where $\partial J(\theta)/\partial \boldsymbol{a}$ is computed from the critic and $\partial \boldsymbol{a}/\partial \theta$ is computed from the policy network. We differentiate through the value network into the policy network. More formally,

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\nabla_\theta \pi_\theta(\boldsymbol{o}) \mathbb{E}\left[\nabla_{\boldsymbol{a}} \boldsymbol{Z}_\phi(\boldsymbol{o}, \boldsymbol{a})\right]|_{\boldsymbol{a}=\pi_\theta(\boldsymbol{o})}\right] \tag{8}$$

describes how the policy parameters, θ, should be updated to increase policy performance. The policy parameters are then adjusted in the direction of the gradients calculated in Eq. (8) a small amount, called the learning rate, α:

$$\theta = \theta + \nabla_\theta J(\theta)\alpha \tag{9}$$

During data collection, K parallel agents run simulations using the most up-to-date version of the policy. To encourage exploration of the action space and environment, it is important to not simply follow the policy's output—better actions may exist. While there are many methods for forcing exploration, this work adds Gaussian noise to the action calculated by the policy. The action taken at time step t is therefore

$$\boldsymbol{a}_t = \pi_\theta(\boldsymbol{o}_t) + \mathcal{N}(0, \sigma^2) \tag{10}$$

Algorithm 1. D4PG (Barth-Maron et al., 2018)

Learner

Initialize policy network weights θ and value network weights ϕ randomly
Initialize smoothed policy and target value network weights $\theta' = \theta$ and $\phi' = \phi$
Launch K actors and copy policy weights θ to each actor
repeat
  Sample a batch of M data points from the replay buffer
  Compute the target value distribution used to train the value network $\boldsymbol{Y}_t = \sum_{n=0}^{N-1} \gamma^n r_{t+n} + \gamma^N \boldsymbol{Z}_{\phi'}(\boldsymbol{o}_{t+N}, \pi_{\theta'}(\boldsymbol{o}_{t+N}))$
  Update value network weights ϕ by minimizing the loss function $L(\phi) = \mathbb{E}[-\boldsymbol{Y}_t \log(\boldsymbol{Z}_\phi(\boldsymbol{o}_t, \boldsymbol{a}_t))]$ using learning rate β
  Compute policy gradients using $\nabla_\theta J(\theta) = \mathbb{E}\left[\nabla_\theta \pi_\theta(\boldsymbol{o}_t) \mathbb{E}\left[\nabla_{\boldsymbol{a}} \boldsymbol{Z}_\phi(\boldsymbol{o}_t, \boldsymbol{a}_t)\right]|_{\boldsymbol{a}=\pi_\theta(\boldsymbol{o}_t)}\right]$ and update policy weights via $\theta = \theta + \nabla_\theta J(\theta)\alpha$
  Update the smoothed network weights slowly in the direction of the main policy and value network weights $\theta' = (1-\epsilon)\theta' + \epsilon\theta$ and $\phi' = (1-\epsilon)\phi' + \epsilon\phi$ for $\epsilon \ll 1$
until *acceptable performance*

Actor

repeat
  From the given observation, use the policy to obtain an action and add exploration noise $\boldsymbol{a}_t = \pi_\theta(\boldsymbol{o}_t) + \mathcal{N}(0, \sigma^2)$
  Step environment forward one time step using action $\boldsymbol{a}_t$
  Record $(\boldsymbol{o}_t, \boldsymbol{a}_t, r_t = \sum_{n=0}^{N-1} \gamma^n r_{t+n}, \boldsymbol{o}_{t+N})$ and store in the replay buffer R
  At the end of each episode, obtain the most up-to-date version of the policy π_θ from the Learner and reset the environment.
until *acceptable performance*

where $\mathcal{N}(0, \sigma^2)$ is the normal distribution with zero mean and σ standard deviation. Trial and error is the mechanism for the discovery of good actions, as dictated by the reward function; the learning algorithm reinforces those good actions into the policy. A summary of the D4PG algorithm is presented in Algorithm 1.

Appendix B. On-board Hardware

Table 2. General characteristics for the Tawaki autopilot board.

Item	Details
MCU	STM32F7
IMU	ICM20600 (accel, gyro) + LIS3MDL (mag)
Baro	BMP3
Serial	3 UARTS, I2C (5V + 3.3V), SPI
Servo	8 PWM/DShot output (+ ESC telemetry)
RC	2 inputs: PPM, SBUS, Spektrum
AUX	8 multipurpose auxiliary pins (ADC, timers, UART, flow control, GPIO, ...)
Logger	SD card slot
USB	DFU flash, mass storage, serial over USB
Power	6V to 17V input (2-4S LiPo) 3.3V and 5V, 4A output
Weight	12 grams

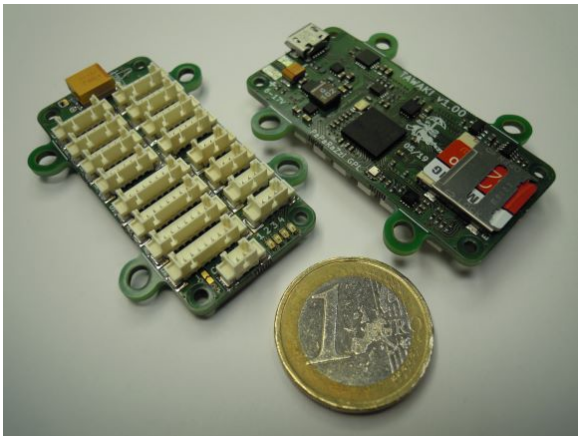


Figure 20. Tawaki autopilot board.