Neural Computing and Applications
https://doi.org/10.1007/s00521-020-05097-x

ORIGINAL ARTICLE

Deep reinforcement learning for drone navigation using sensor data

Victoria J. Hodge[1] · Richard Hawkins[1] · Rob Alexander[1]

Received: 26 November 2019 / Accepted: 4 June 2020

Abstract
Mobile robots such as unmanned aerial vehicles (drones) can be used for surveillance, monitoring and data collection in buildings, infrastructure and environments. The importance of accurate and multifaceted monitoring is well known to identify problems early and prevent them escalating. This motivates the need for flexible, autonomous and powerful decision-making mobile robots. These systems need to be able to learn through fusing data from multiple sources. Until very recently, they have been task specific. In this paper, we describe a generic navigation algorithm that uses data from sensors on-board the drone to guide the drone to the site of the problem. In hazardous and safety-critical situations, locating problems accurately and rapidly is vital. We use the proximal policy optimisation deep reinforcement learning algorithm coupled with incremental curriculum learning and long short-term memory neural networks to implement our generic and adaptable navigation algorithm. We evaluate different configurations against a heuristic technique to demonstrate its accuracy and efficiency. Finally, we consider how safety of the drone could be assured by assessing how safely the drone would perform using our navigation algorithm in real-world scenarios.

Keywords UAV · drone · Deep reinforcement learning · Deep neural network · Navigation · Safety assurance

1 Introduction

Rapid and accurate sensor analysis has many applications relevant to society today (see for example, [2, 41]). These include the detection and identification of chemical leaks, gas leaks, forest fires, disaster monitoring and search and rescue. There are other less dramatic applications such as agricultural, construction and environmental monitoring. The sensors take measurements of specific chemical concentrations or infrared or thermal imaging levels which can then be analysed to detect anomalies [22]. In this context, we define an anomaly as an outlying observation that appears to deviate markedly from other members of the sample in which it occurs [6], i.e. a sensor reading that appears to be inconsistent with the remainder of the set of sensor readings [6]. In the sensor monitoring application domain, an anomaly is indicative of a problem that needs investigating further [21] such as a gas leak where the gas reading detected by the sensors is elevated above normal background readings for that particular gas. Sensor monitoring for environments, infrastructure and buildings needs to be mobile, flexible, robust and have the ability to be used in a broad range of environments. Many current sensors, including IoT sensor systems, are static with fixed mountings. One solution to the mobility and flexibility issues is to mount the sensors on robotic/autonomous systems such as unmanned aerial vehicles (UAVs) often referred to as drones [3]. These drones can be used autonomously or operated by a human drone pilot to detect and locate anomalies or perform search and rescue. They can operate in areas unsafe or inaccessible to humans. Example applications of sensor drones for condition monitoring include agricultural analysis [39], construction inspection [25], environmental (ecological) monitoring [3, 28], wildlife monitoring [16], disaster analysis [15], forest fire monitoring [12], gas detection [36, 42] and search and rescue [17, 43, 51].

✉ Victoria J. Hodge
victoria.hodge@york.ac.uk

Richard Hawkins
richard.hawkins@york.ac.uk

Rob Alexander
rob.alexander@york.ac.uk

[1] Department of Computer Science, University of York, York YO10 5GH, UK

Published online: 21 June 2020

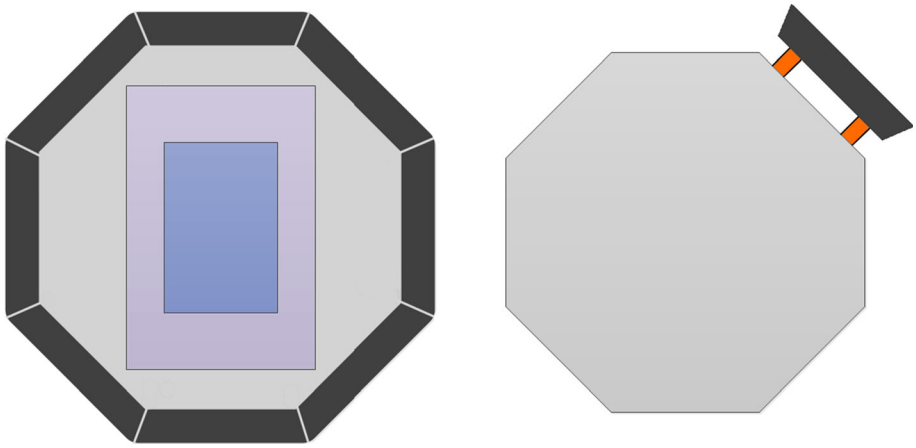

Fig. 1 Schematic of a possible sensor module which attaches underneath the drone. Eight sensor plates are shown in black and clip together in an octagon using magnets or clips. The sensors are arranged facing outwards to face 8 directions. This octagon then clips to the processing board (shown in purple) using magnets or clips (colour figure online)

In this paper, we present a drone navigation recommender system for small or microdrones [3] although it can easily be used on other applications including larger drones or unmanned ground vehicles (UGVs). It can be considered analogous to a sat-nav system in a car in terms of operation and visuals—it recommends the best direction of travel to get to the goal location (anomaly site). Hilder et al. [19] documented a system for UGVs that had sensors mounted on board. The system used artificial immune techniques to detect anomalies in the sensor data, but the ground vehicle used random walk to find the target. Here, we use intelligent guidance incorporating the sensor data to guide the drone to the anomaly site.

The following research topics are not in the scope of this paper: UAV sensor module development and sensor anomaly detection. We assume a sensor module attaches underneath the drone: for example, to a gimbal or using a mounting bracket. The gimbal provides geometric stability for the sensors. This attachment approach has been widely used in drone remote sensing [3]. We envisage this module (see Fig. 1 for an example) containing a number of sensors arranged in formation around a processing plate containing a processing board such as a Raspberry Pi[1] for lightweight processing of simple sensor data, a Nvidia Jetson Nano[2] for heavier data processing such as image sensor data or bigger boards such as Intel Nuc[3] or Nvidia Jetson[4] if the drone's payload permits and more heavyweight processing is needed. Thus, the processing board can collect the data from the sensors. The sensors may be any arrangement of sensor types appropriate for the anomaly detection application. Sensor types range from the simplest temperature and humidity sensors to high-end thermal imaging and camera sensors. These sensor data are combined with location data and obstacle detection data from a collision avoidance mechanism (such as the drone's mechanism) to enable anomaly detection and navigation.

Every time the drone's sensor module collects sensor data, these data can be processed for anomalies on-board (in the module's processing plate) if sufficient compute capacity is available or transmitted to a nearby device to process [24]. The anomaly detection would be a two-stage process: (1) determine whether there is an anomaly and (2) determine which sensor is giving the most anomalous reading. The anomaly detection software will use a real-time sensor data anomaly detector for step 1 such as those analysed in [23].

In this paper, we focus on the recommender software. It uses artificial intelligence (AI) and operates once this anomaly detection software detects an anomaly [22]. The sensor data are coupled with the drone's current direction obtained either via the drone's on-board navigation system or from a compass mounted with the sensors and the obstacle detection data from the drone's collision avoidance mechanism. The recommender uses these data as input to an off-policy deep learning model to recommend the direction of travel for the drone according to the current prevailing conditions, surroundings and sensor readings. This deep learning AI guides the drone to the site of the anomaly and the drone can transmit the exact anomaly site coordinates (and sensor data if needed) back to base for further investigation as appropriate. This is particularly important for safety-critical incidents or where the investigators have to wear hazardous material suits or breathing apparatus with only a very limited time of usage (often around 20 minutes). They can head straight to the tagged location while the drone performs further sensor analyses. The AI recommender allows the human pilot and on-board collision avoidance or the drone's autonomous navigation system (including collision avoidance) to focus on the actual navigation and the collision avoidance. This latter

[1] https://www.raspberrypi.org/.

[2] https://developer.nvidia.com/buy-jetson.

[3] https://www.intel.co.uk/content/www/uk/en/products/boards-kits/nuc/boards.html.

[4] https://developer.nvidia.com/buy-jetson.

mechanism provides a separate safety net which overrides the AI automatically if the AI recommendation would lead the drone into a dangerous situation (such as a collision with a concrete pillar).

Previous work used AI for drone navigation, processing the images from on-board cameras for wayfinding and collision avoidance [9, 43, 52]. However, this paper introduces a new direction recommender to work in conjunction with the navigator (human or AI pilot). Our recommender AI needs to be able to navigate generic environments, navigate novel environments that it has not seen before and navigate using only minimal information available from a drone and the sensors mounted on-board. We know: the drone's location (generally from its on-board GPS), if there is an obstacle to the immediate N, E, S, W of the drone and the direction and magnitude of the sensor reading.

Current navigation algorithms can be subdivided into:

- those that use global data (have an overview of the entire navigation space)
- those that use only local (partially observable) information.

A global algorithm such as A* needs visibility of the whole exploration space (the whole grid). It expands paths to determine the best path and backtracks if a path is no longer best; it expands the search tree and must examine all equally meritorious paths to find the optimal path as shown in Fig. 2. Hence, the drone would have to fly paths and backtrack to explore. A* often needs to examine large areas of complex environments particularly if there are concave obstacles. Additionally, A* cannot cope with dynamic environments or next state transitions that are stochastic. If the grid layout changes between moves or the drone is blown off course, then A* needs to recalculate from first principles.

We need to navigate with only incomplete (partially observable) information examining the drone's local area. The drone may also need to navigate potentially changing and hazardous environments. Our navigator described in this paper uses a partially observable step-by-step approach with potential for recalculation at each step. In this paper, we focus on static environments as a first step in developing our drone navigation system. Although we do not explore dynamic environments in this paper, we need an approach that can cope with changing layouts as we intend to develop our algorithm to navigate dynamic environments as well in the future.

The contributions of this paper are:

- A novel recommender system for drone navigation combining sensor data with AI and requiring only minimal information. Hilder et al. [19] used random walk for a similar system for UGVs (buggies) but that can get stuck inside obstacles as we show in our evaluation in Sect. 5.
- We combine two deep learning techniques, (1) proximal policy optimisation (PPO) [45] for deep reinforcement learning to learn navigation using minimal information with (2) long short-term memory networks (LSTMs) [20] to provide navigation memory to overcome obstacles.
- We evaluate our system in a simulation environment to allow us to easily and thoroughly test it before transferring to real-world testing which is more difficult logistically and very expensive. Once we establish the merits and limits of the system within the simulation environment, we can deploy it in real-world settings and continue the optimisation.
- We define safety requirements for the system using a systematic functional failure analysis (FFA) [40]. We consider each system function in turn and use standard guidewords to consider deviations in those functions (i.e. function not provided, function provided when not required, function provided incorrectly). We then identify the potential worst-case effects of each functional deviation and hence identify a set of safety requirements for the system.

In Sect. 2 we analyse potential algorithms, we describe deep reinforcement learning and why we are using it here, Sect. 3 describes how we implement a drone navigation simulation using sensor data coupled with deep reinforcement learning to guide the drone, Sect. 4 gives a brief overview of the simulation's operation, and we evaluate the drone navigation AI in Sect. 5. Section 6 provides a safety assurance assessment of our system and identifies a set of safety requirements. We discuss our evaluations in Sect. 7 and provide conclusions and further work possibilities in Sect. 8.

2 Reinforcement learning (RL)

As stated above, we use a local algorithm to navigate as the drone only has local visibility of the exploration spaces (they are partially observable). There are a number of local navigation approaches. Genetic algorithms can perform partially observable navigation [13]. They generate a population of randomly generated solutions and use the principles of natural selection to select useful sets of solutions. However, they have a tendency to get stuck in local minima. Fuzzy logic algorithms [55] have been used to learn to navigate, and Aouf et al. [4] demonstrated that their fuzzy logic approach outperformed three meta-heuristic (swarm intelligence) algorithms: particle swarm

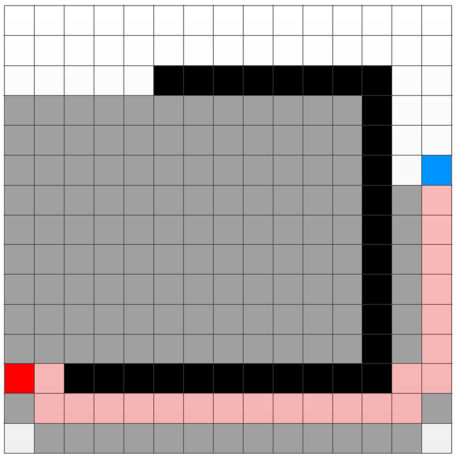

Fig. 2 The grid cells examined during A* search from the red square to the blue square. The black squares are an obstacle. The algorithm finds the optimal path from the red square to the blue square but explores a large portion of the grid. The drone would have to fly across all of the grey squares and the pink squares to find the path to the blue square (colour figure online)

optimisation, artificial bee colony and a meta-heuristic Firefly algorithm, for navigation time and path length. However, fuzzy logic algorithms struggle in dynamic environments as they are too slow to recompute the path on the fly when the environment changes [46]. Patle et al. [38] review a number of techniques including these meta-heuristic algorithms such as GAs and swarm intelligence (including particle swarm optimisation, artificial bee colony, firefly algorithm and ant colony optimisation) for robot navigation. Meta-heuristic swarm intelligence algorithms aim to find the best model by exploring the model space in an intelligent manner via emergent behaviour. They aim to find a good rather than optimal solution and can also become trapped in local minima. Patle et al. [38] conclude that GAs and swarm intelligence can navigate in uncertain environments, but they are complex and not suitable for low-cost robots. Regular neural networks such as multilayer perceptrons can be used to train a navigation model [38, 46], but they do not have the computational power of deep learning algorithms and would be restricted to simpler environments. Navigation algorithms can use deep classification learning with deep neural networks. The deep neural network learns to navigate by generating labelled training data where the label scores the quality of the path chosen [49]. However, it is both time-consuming and difficult to accurately label a large enough set of training examples.

In contrast, deep reinforcement learning (deep RL) uses a trial and error approach which generates rewards and penalties as the drone navigates. A key aim of this deep RL is producing adaptive systems capable of experience-driven learning in the real world. Matiisen et al. [34] observed that deep RL has been used to solve difficult tasks in video games [35], locomotion [33, 44] and robotics [31]. It has also been used in robot navigation [56] where the authors could navigate 20×20 grids with 62% success rate.

Reinforcement learning (RL) itself is an autonomous mathematical framework for experience-driven learning [5]. As noted by Arulkumaran et al. [5], RL has had some success previously such as helicopter navigation [37], but these approaches are not generic, scalable and are limited to relatively simple challenges.

Formally, RL is a Markov decision process (MDP) as shown in Fig. 3, comprising:

- A finite set of states S, plus a distribution of starting states $p(s_0)$. There may be a terminal state, s_T. The complexity of the learning task is exponential with respect to the number of variables used to define a state. We later describe how we minimise the state representation.
- A set of actions A covering all agents, available in each state. We have only one agent which can move in one of four possible directions.
- Transition dynamics (policy) $\pi_\theta(s_{t+1}|s_t, a_t)$ that map a state/action pair at time t onto a distribution of states at time $t+1$ using parameter set θ. Transitions only depend on the current state and action (Markov assumption)
- An instantaneous reward function $R(s_t, a_t, s_{t+1})$ associated with each transition; used to assess the optimum transition.
- A discount factor $\gamma \in [0, 1]$, which is the current value of future rewards. It quantifies the difference in importance between immediate rewards and future rewards (lower values place more emphasis on immediate rewards).
- Memorylessness. Once the current state is known, the history is erased as the current Markov state contains all useful information from the history; "the future is independent of the past given the present". **This will**

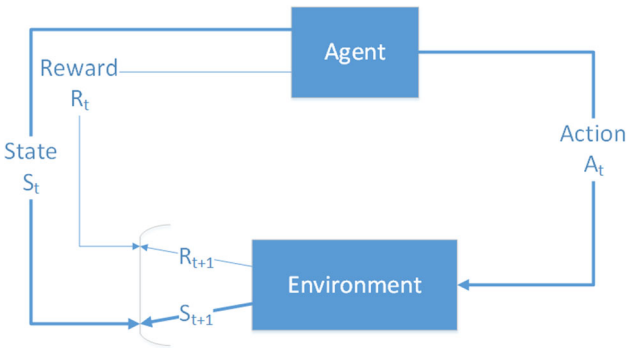

Fig. 3 The agent–environment framework of a Markov decision process

prove important later on as we develop our recommender system.

We treat our drone navigation problem analogously to the Grid-World navigation problem [48]. In this paper, we know the drone's GPS location, what is to the immediate N, E, S, W of the drone and the direction of the sensor reading in Cartesian coordinates (x-distance, y-distance) (where N, E, S and W are relative to the ground in this example). In a real-world scenario, we may know the direction and magnitude of the sensor readings in polar coordinates using direction relative to the ground or relative to the drone as appropriate. Magnitude and direction can easily be converted from polar coordinates to Cartesian coordinates. If there are 8 sensors arranged in an octagon as shown in Fig. 4, then the highest sensor reading gives the direction to fly relative to ground or drone, and the magnitude (strength) of the anomaly. Here the drone would head north-east.

In our drone navigation recommender system, only part of the environment is observable at any point in time. In the real world, we would only know the immediate vicinity of the drone via the drone's collision avoidance mechanism. We do not know the locations of obstacles that lie further ahead (in the real world they may be obscured by closer obstacles) or beyond the range of the drone's vision. Alternative formulations for the Grid-World navigation problem treat the environment as a picture (observations) where each cell of the grid maps to a pixel whose value represents the contents of that cell {empty, obstacle, goal} [50]. This requires complete information of the environment which is not available from the drone. Additionally, this approach does not scale to different grid sizes as it learns to navigate using $N \times N$ grids as images. A deep learner trained on a 16×16 observations grid cannot generalise to a 32×32 grid using this observation formulation as the network input size would be different (16×16 compared to 32×32) and would be misaligned. Our formulation is scalable, adaptable and flexible.

Fig. 4 The eight sensor plates clip together in an octagon formation. Our system uses anomaly detection to determine whether the sensor readings are outside the normal range for the environment. If they are abnormal, then it detects which sensor is giving the most anomalous reading. Here, the north-east sensor is most anomalous and indicates the direction to head

2.1 Partially observable MDPs (POMDPs)

To train the drone using this partial (local) information, we use a generalisation of MDPs known as partially observable MDPs (POMDPs). In a POMDP, the agent receives an observation $o_t \in \Omega$, where an observation's distribution $p(o_{t+1}|s_{t+1}, a_t)$ depends on the previous action a_t and the current state s_{t+1}. Here, the observation is described by a mapping in a state-space model that depends on the current state {sensor direction, sensor magnitude, N, E, S, W space} and the previously applied action (whether the drone moved N, E, S, W).

An MDP represents transition probabilities from state to state. A policy π is a distribution over actions given states, $\pi_\theta(a_t|s_t) = P[A_t = a_t|S_t = s_t]$. A policy fully defines the behaviour of an agent given the current state s_t; it generates an action a_t given the current state s_t and that action, when executed, generates a reward r_t. The aim of RL is to identify the optimal policy, π^*, which maximises the reward over all states (i.e. maximises the expected reward value E): $\pi^* = \text{argmax}_\pi E[R_t|\pi]$. There are two common approaches for determining the optimal policy: *value learning* which maintains a value function model, and *policy learning* which is model free and searches directly for the optimal policy. In this paper, we use policy learning. Value learning considers all actions at each iteration and is slow; it takes $|A|$ times longer than policy evaluation. Also, the policy does not change at each iteration wasting further time.

2.2 Policy gradients learning

The drawbacks of RL, as with many other AI algorithms, are memory usage, computational complexity, and sample complexity. There has recently been a move to underpin RL with deep neural networks (DNNs) which provide powerful function approximation and representation learning properties. One subset of deep RL algorithms are Policy gradient algorithms. They search for a local maximum in the policy quality by ascending the gradient of the policy. Policy gradients learning is robust, but the gradient variance is high. To lower this variance, we use unbiased estimates of the gradient and subtract the average return over several episodes which acts as a baseline. Additionally, policy gradients have a large parameter set which can create severe local minima. To minimise the likelihood of local minima, we can use trust regions. Trust region search constrains the optimisation steps so that they lie within a region where the true cost function approximation still

holds. We can reduce the likelihood of a very poor update by ensuring that updated policies have low deviations from prior policies, by using the Kullback–Leibler (KL) divergence [29] to measure the deviation between the current and proposed policy. Trust region policy optimisation (TRPO) has demonstrated robustness by limiting the amount the policy can change and guaranteeing that it is monotonically improving. However, this constrained optimisation requires calculating second order gradients, limiting its applicability. To overcome this, Schulman et al. [45] developed the proximal policy optimisation (PPO) algorithm which performs unconstrained optimisation, requiring only first-order gradient information. PPO executes multiple epochs of stochastic gradient descent to perform each policy update. Hence, it performs a trust region update in a way that is compatible with stochastic gradient descent, thus simplifying the algorithm by removing the need to make adaptive updates. PPO computes an update at each time step t that minimises the cost function while ensuring the deviation from the previous policy is relatively small.

2.3 Proximal policy optimisation (PPO) algorithm

PPO interleaves policy optimisation with collecting new examples, in our case running a navigation example. PPO demonstrates performance comparable to or better than state-of-the-art approaches but is much simpler to implement and tune [45]. Its performance has also been demonstrated for Minecraft and simple maze environments [7]. PPO optimises the KL penalty by forming a lower bound using a clipped importance ratio $L^{\text{Clip}}(\theta)$; this is an element-wise minimum between the clipped and unclipped objective. PPO performs optimisation using a batch of navigation examples and minibatch stochastic gradient descent to maximise the objective. This simplifies the algorithm by removing the KL penalty and the requirement to make adaptive updates. It finds a reliable update between the updated policy π and the old policy π_{old} which generated the batch of navigation examples. It prevents PPO from being too greedy and trying to update too much at once and updating outside the region where this sample offers a good approximation. PPO is described in Eq. 1, θ is the policy parameter, $\hat{E}_t$ is the empirical expectation over time, ratio$_t$ is the ratio of the probability of the new and old policies, $\hat{A}_t$ is the estimated advantage at time t and ϵ is a hyper-parameter set to 0.1 or 0.2 which is an entropy factor to boost exploration.

$$L^{\text{Clip}}(\theta) = \hat{E}_t[\min(\frac{\pi(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)})\hat{A}_t, \text{clip}\left(\frac{\pi(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}), 1-\epsilon, 1+\epsilon\right)\hat{A}_t)] \quad (1)$$

PPO is gaining popularity for a range of RL tasks as it is less expensive whilst retaining the performance of TRPO [45].

3 Models and system architecture

Our drone simulation uses Unity 3-D's ML-agents framework [26] to design, develop and test the simulations prior to real-world deployment. ML-agents uses the Unity 3-D C# development framework as a front-end and middleware interfacing to a Google TensorFlow [1] backend in Python. The MS Windows version of ML-agents has a DLL library to interface between C# and Python. It allows users to develop environments for training intelligent agents [26].

In this paper, we focus on 2-D navigation and do not consider the altitude of the drone. Thus, our anomaly detection problem is a deterministic, single-agent search, POMDP problem implemented using Grid-World in Unity 3-D ML-agents. We specify the grid size and number of obstacles and the grid is randomly generated (see Fig. 5 for an example grid). The Unity 3-D game environment runs the simulation. Our system comprises three main interacting processes.

3.1 Agents

In the ML-agents framework, the agents are Unity 3-D Game Objects as demonstrated in [10, 11, 32] and [54]. In our simulation, the agent is a drone. In ML-agents, the agent generates the state, performs the prescribed actions and assigns the cumulative rewards. The agent is linked to exactly one brain (Sect. 3.2).

The agent (drone) takes the prescribed actions (move N, E, S or W by one cell) to navigate the grid, avoiding the obstacles and finding the goal. Our state space is a length 6 vector of the contents of the adjacent grid cells (N, E, S, W), and the x-distance and y-distance to the target (anomaly). This is both compact, scalable and realistically achieved. (In the real world, the drone can only sense its local environment; it cannot guarantee to see ahead due to occlusions.)

In our RL, the agent receives a small penalty for each movement, a positive reward (+ 1) for reaching the goal, and a negative reward (− 1) for colliding with an obstacle.

3.2 Brains

Each agent has one brain linked to it which provides the intelligence and determines the actions. The brain provides the logic for making decisions for the agent; it determines the best action to take at each instance. Our brain uses the

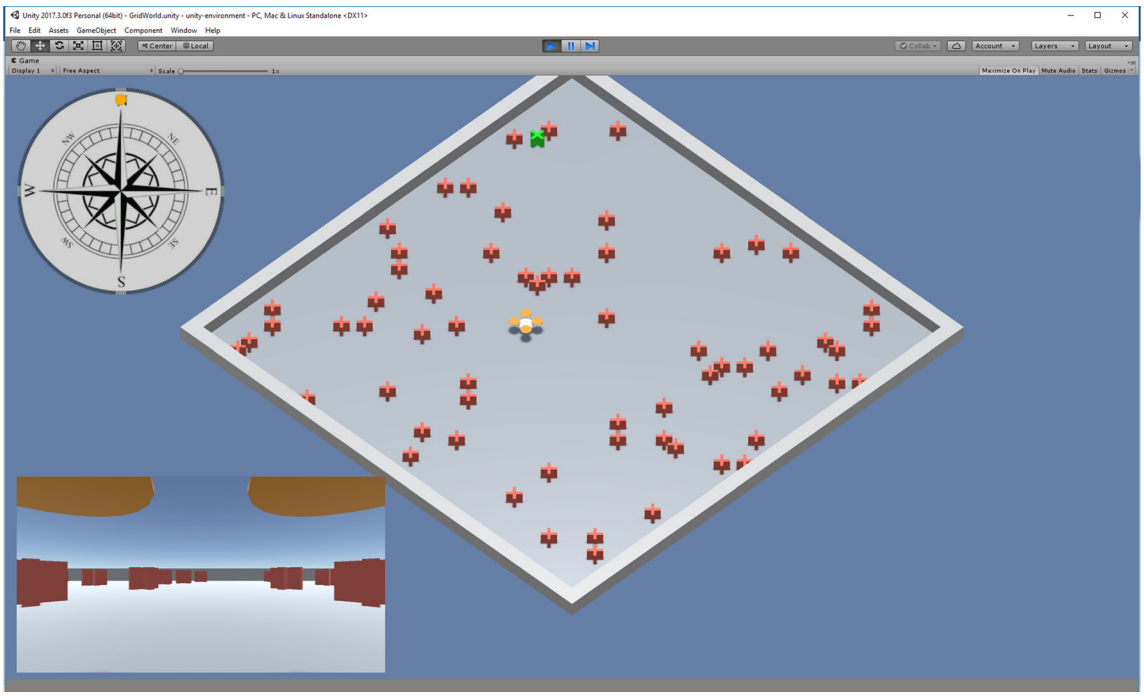

Fig. 5 A randomly generated Unity 3-D ML-agents Grid-World with a 32 × 32 grid, 64 obstacles (red ×) and one goal (green +). The AI navigates the drone to the goal. The compass (top left) shows the recommended direction of travel to the pilot. The inset bottom left is what the drone's forward-facing camera would see (colour figure online)

proximal policy optimisation (PPO) RL algorithm as developed by OpenAI [45] which is optimised for real-time environments. The ML-agents' PPO algorithm is implemented in TensorFlow and run in a separate Python process (communicating with the running Unity application over a socket).

The PPO algorithm receives the state-space representation (the contents of the adjacent grid cells (N, E, S, W), and the x-distance and y-distance) and the set of possible actions (move N, E, S or W one cell) as input and selects the action to maximise the reward using the learned policy.

3.3 Academy

This element of the environment orchestrates the decision-making process. It forms a conduit between the brain (logic) and the actual Python TensorFlow implementation of the brain which programmatically contains the logic as a learned deep neural network model.

3.4 Configuration

To configure the agent and brain, we spent a long time evaluating different agent, state, reward configurations. These settings are key to a successful implementation so it is worth investing time evaluating the different configurations. We analysed:

- Different state representations, in particular different distance representations where we used different scaling factors relating to the grid size and the remaining distance. We found the best results came from using a state space of N, E, S, W, $\mathrm{d}(x)$, $\mathrm{d}(y)$ where $\mathrm{d}(x) = \frac{\mathrm{dist}_x}{\max(\mathrm{dist}_x, \mathrm{dist}_y)}$ and $\mathrm{d}(y) = \frac{\mathrm{dist}_y}{\max(\mathrm{dist}_x, \mathrm{dist}_y)}$
- Different step rewards where we used different scaling factors relating to the grid size finding a step penalty of $\mathrm{stepPenalty} = \frac{-1}{(\mathrm{longestPath})}$ where $\mathrm{longestPath} = ((\mathrm{gridSize} - 1) * \mathrm{gridSize}/2) + \mathrm{gridSize})$ was best.
- A number of PPO hyper-parameters sets and found the best results came from the settings listed in "Appendix".

If we train the agent using only 1 obstacle in the training grids, then the agent learns to travel directly to the goal which is desirable. However, after training and during evaluation, it struggles when it encounters more complex obstacles (2 or more red crosses joined). If we commence training with multiple obstacles, e.g. 32 obstacles (randomly placed red crosses), then it learns to walk haphazardly. This enables it to overcome more complex obstacles, but it does not travel directly to the goal when the Grid-World environment has very few obstacles. This is problematic and nullifies our desire for the agent to be generic and able to tackle a variety of environments. Hence, we

looked at step-by-step (curriculum) learning described next.

3.5 Curriculum learning

Training of deep learning networks can be expedited by exploiting knowledge learned from previous related tasks. There are several techniques including: transfer learning, multitask learning and curriculum learning [8]. We focus on curriculum learning because it can begin learning in simulators with visual rendering and then the learned model can be fine-tuned in real-world applications. This is beneficial to our application. It allows us to use drone simulations to bootstrap the system and progress to drone flights. Formally, a curriculum is a series of lessons (sequence of training criteria). Curriculum learning starts with a simple task and gradually increases the complexity of the task as learning progresses until we reach the training criterion of interest. It does not forget previously learned instances. Each lesson (training criterion) generates a different set of weights during training building on the previous weights. We progress from 1 obstacle to 4 then 8 then 16 then 32 all in a 16×16 grid. At the end of the sequence, the drone can still efficiently navigate the 16×16 grid with 1 obstacle as the network has not forgotten the knowledge gained during the first lesson. It can also now efficiently navigate the 16×16 grid with 32 obstacles from the knowledge gained during the final lesson.

In this paper, we use an adaptive curriculum learning approach that we call "incremental curriculum learning". Curriculum learning requires the number of iterations for each lesson to be pre-specified, e.g. train lesson one for 5 million iterations. Often, this number cannot be determined accurately in advance. Incremental curriculum learning allows the user to adapt the number of iterations for each lesson to optimise training. It trains the network during a curriculum lesson for the pre-specified number of iterations unless a user stops the learning process early if the model is trained sufficiently or the user adds extra iterations if the model is not sufficiently trained after the pre-specified number of iterations. There are a number of metrics we can use to determine how many epochs to train each lesson such as loss, entropy or mean final reward. We analysed the metrics and found that mean final reward generated the best model for navigation with our incremental curriculum learning. The other metrics tended to either over-train or under-train the models leading to poor generalisation capabilities. Thus, we use the mean final reward to identify when each lesson should end. For example, if we specify train lesson one for 5 million iterations and examine the agent's mean final reward (averaged over every 10,000 training iterations), we can determine if the reward is still increasing or has become stable. If it is still increasing, then we assume the agent has not learned this curriculum step sufficiently and can add a further 0.5 million iterations to lesson one and test again once it has run 5.5 million iterations. Hence, we incrementally learn each lesson until we are satisfied that the PPO has learned that lesson sufficiently then we move onto the next lesson (more complex task in the curriculum). We analyse this incremental curriculum learning further in the Evaluations in Sect. 5.

3.6 Memory

In Sect. 2, we formally defined an MDP. We highlighted that MDPs are memoryless. This proved an issue for our navigation recommender system. When the agent encounters concave obstacles (cul-de-sacs) a lack of memory is a problem. The PPO agent steps back and forth or circles repeatedly as it cannot remember previous movements. To overcome this, we added an LSTM memory layer for the PPO deep learning, as shown in Fig. 6. An LSTM is a gated recurrent neural network [20] capable of learning longer-term dependencies (sequences) making it ideal to provide a memory layer for the agent.

An LSTM network comprises memory blocks called cells. The cells form sequences and are responsible for remembering and memory manipulations that update the hidden state (memory). Each cell is gated so it can store or delete information (by opening and closing the gate). Thus, the LSTM can read, write and delete information from its memory. The sequence length is the number of steps the agent must remember. It is also the length of input data and the sequences passed through the LSTM during training. This length needs to be sufficiently long to capture the

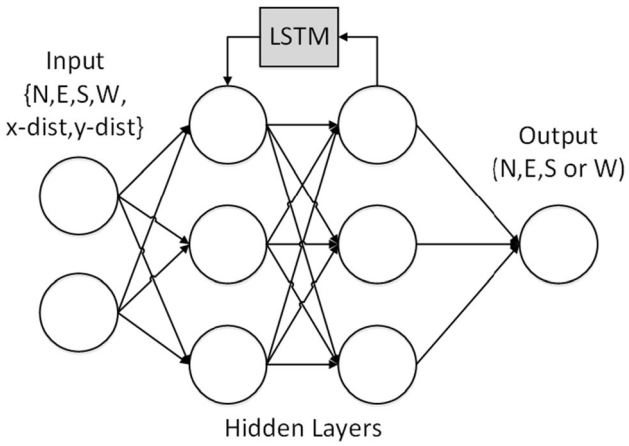

Fig. 6 Schematic of the PPO and LSTM network. There are 2 hidden layers in our PPO network with 64 nodes per layer. The LSTM provides a recurrent connection between hidden layers. This loop back allows the network "to remember" the previous inputs and to include this recurrent information into the decision-making. Further hyper-parameter settings are given in the "Appendix"

information the agent must remember. However, a longer sequence increases the training time as it increases the LSTM complexity. In this case, we need to remember sufficient steps to allow the agent to navigate cul-de-sacs and other more complex obstacles. LSTMs are recurrent and backpropagate the output error through time and layers. This recurrent mechanism allows such networks to learn over time steps.

3.7 Training

During training, the TensorFlow PPO model with LSTM sequence memory performs the decision-making. The TensorFlow model is separate from the Unity environment and communicates via a socket. We trained the brain for 50 million training episodes using our incremental curriculum learning. Each episode is generated independently by Unity 3-D as a separate navigation task. For each grid (episode), the navigator either solves the grid, fails or times out. It then moves on to the next grid layout. The obstacles are placed in the grid using the Unity 3-D C# random number generator to select the positions.

The PPO network setup is shown in Fig. 6 and the set of parameters for Unity ML-Agents is given in the "Appendix". Input to the network is the six-dimensional state vector (N, E, S, W, x-distance, y-distance) and it outputs which action to take: move N, E, S or W by one step.

4 Simulation operation

Once we have a trained model, we switch to Internal mode where the Unity 3-D environment uses it to navigate. Unity passes the agent's current state to the stored TensorFlow graph which returns the recommended action. This represents the best action to take given the current state of the system and the set of possible actions. In Internal mode, no more learning is performed and the model graph is frozen. We can train the model further by switching back to training mode in the Unity 3-D setup if needed.

5 Evaluations

Our first analysis is to investigate our incremental curriculum learning. We demonstrate why we use the incremental approach and how we use the training reward metric to determine when to stop training each lesson. We evaluate two versions of the drone AI and a baseline PPO without memory.

- The baseline PPO has no LSTM memory but trained with incremental curriculum learning (PPO).
- The first drone AI uses PPO trained with curriculum learning and having an LSTM with memory length 8 (it remembers the last 8 steps taken) (PPO_8).
- The second drone AI is identical except the memory is length 16 (PPO_{16}).

Figure 7 shows the mean reward while training 5,000,000 iterations of the first lesson of the curriculum (16×16 grid with 1 obstacle) for PPO, PPO_8 and PPO_{16} along with the mean reward during training of the second lesson of the curriculum (16×16 grid with 4 obstacles) for PPO_8_L2. PPO_8_L2 in the second lesson has already undergone 5,000,000 training iterations on a 16×16 grid with 1 obstacle during lesson 1 and we show how this affects training. The lower chart is a zoomed version of the top chart and shows the oscillations in the mean reward more clearly. Similarly, Fig. 8 shows the standard deviation of the reward during training of the first lesson of the curriculum for PPO, PPO_8 and PPO_{16} along with the reward standard deviation during training of the second lesson of the curriculum for PPO_8.

Figures 7 and 8 show that for lesson 1, PPO with no memory initially learns fastest as the mean reward and reward standard deviation plot lines oscillate least and settle quickest but there is a slight increase in oscillation around 4 million training iterations. The figures show that the larger the AI memory then the longer the AI takes to learn. This is illustrated by the plot line oscillating more and settling slowest initially. At the start of training, PPO_{16} takes 240,000 iteration to reach a mean reward of 0.9 compared to 50,000 for PPO and 150,000 for PPO_8. However, PPO_8 oscillates least after 3 million training iterations as the memory is helping it navigate compared to PPO with no memory. When PPO_8 is on the second lesson, PPO_8_L2 oscillates least of all and settles quickly as it has already learned one previous lesson and carries over its navigation knowledge from one lesson to the next. This variation in length to settle demonstrates why we use incremental curriculum learning as we can vary the length of each lesson according to the AI's time to settle and ensure it undergoes sufficient training to learn. Figures 7 and 8 show that PPO_8 and PPO_8_L2 are ready to move to the next lesson but PPO and PPO_{16} would benefit from at least 0.5 million more iterations.

There is some variation in the standard deviation throughout the lesson due to our random grid generation. We calculate the average reward and reward standard deviation over each block of 10,000 iterations. Some blocks may contain more grids with long paths from the starting point to the goal and other blocks may contain more grids with short paths due to chance. However, the standard deviation should still settle to within a range. Occasionally during learning, the AI may get stuck. This

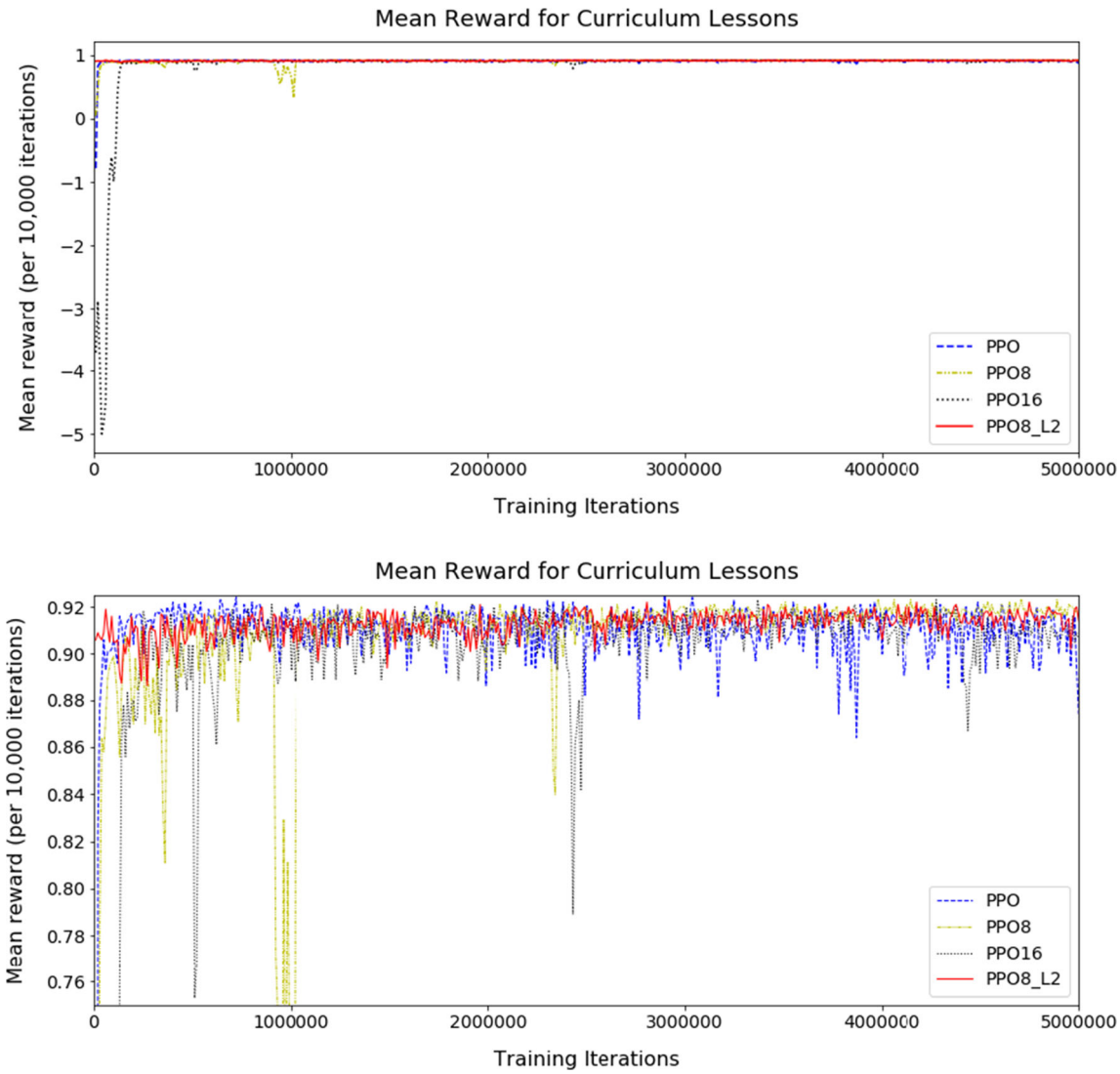

Fig. 7 Line plots of mean reward on the *y*-axis (averaged over each 10,000 iterations) with iteration number on the *x*-axis for PPO, PPO_8 and PPO_{16} on the first lesson of the curriculum (16 × 16 grid with 1 obstacle). PPO_8_L2 is PPO_8 on the second lesson of the curriculum (16 × 16 grid with 4 obstacles). The lower chart is a zoomed version of the top chart

can be seen in the plot for PPO_8 around 1,000,000 iterations where the mean reward drops and the standard deviation increases as the AI has to relearn. Again, by varying lesson length and using a metric, we can ensure the AI has learnt sufficiently before progressing to the next lesson.

Next, we evaluate a baseline PPO without memory, two versions of the drone AI and a simple heuristic approach across a number of Grid-World configurations. The purpose of this evaluation is to show the effectiveness and generalisability of the PPO and LSTM combination and how much benefit using the LSTM to remember the last *N* steps provides for the AI. The Unity 3-D simulator randomly generates 2000 episodes of the Grid-World for each of the different drone AI configurations. Using visual observation, we qualitatively assessed the obstacle layouts in around 50–100 of each of the 2000 episode sets to ensure they were distributed across the grid and provided a good mix of obstacle shapes and sizes that were both close together and further apart. We analyse the episode length (how many steps the drone takes to reach the goal), the reward when the episode ends (either with the drone reaching the goal, hitting an obstacle or running out of steps) and the accuracy (how many times the agent successfully finds the goal in the 2000 runs). Each episode can last up to 1000 steps before it times out. We evaluate:

- PPO_8—the drone AI (PPO with an LSTM with memory length 8).

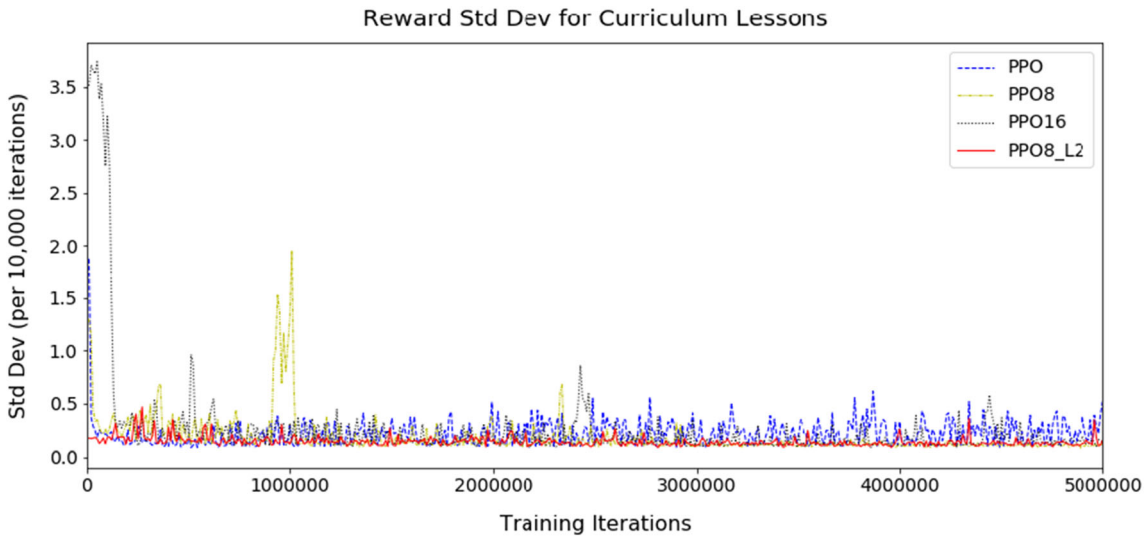

Fig. 8 Line plot of reward standard deviation on the *y*-axis (over each 10,000 iterations) with iteration number on the *x*-axis for PPO, PPO_8 and PPO_{16} on the first lesson of the curriculum (16 × 16 grid with 1 obstacle). PPO_8_4 is PPO_8 on the second lesson of the curriculum (16 × 16 grid with 4 obstacles)

- PPO_{16}—the drone AI (PPO with an LSTM with memory length 16).
- PPO—the baseline PPO with no memory.
- Heuristic—the simple heuristic calculates the distance in both the *x* and *y* directions from the drone's current position to the goal and then moves in the direction (N, E, S, W) with the lowest distance to the goal. This is analogous to PPO without LSTM that uses the *x* and *y* distances and the contents of the four grid cells to the N, E, S, W to determine the next move. If the drone cannot move in the direction recommended by the heuristic due to an obstacle then it randomly selects a direction to move in that is not blocked by an obstacle. The PPO without LSTM tends to get stuck in obstacles but the heuristic's random moves may free it from the obstacle.

Figure 9 and Tables 1 and 2 detail the analyses of the algorithms each across eight Grid-World scenarios with different grid sizes and numbers of obstacles. Note, the agents only trained on the 32 × 32 grid with 32 obstacles; all other Grid-World setups are novel.

For the sensor drone, it is desirable to have low episode length (fewest steps) but high reward (lowest penalties) and the highest accuracy (highest success rate) possible. We want the drone to find the goal and find it in as few steps as possible. It is possible to have low episode length with low reward if the drone takes a few steps and then hits an obstacle which is not desirable. Considering these factors, the PPO with LSTM length 8 performs best overall. The PPO with LSTM length 16 is the best approach for grids size 32 with 256 obstacles which represents a very overcrowded environment filled with obstacles. PPO and the heuristic approaches get stuck in more complex environments and have a higher episode length (step count) overall. However, for the 64 grid with 64 obstacles heuristic is best and PPO is best for 64 grid with 128 and 256 obstacles w.r.t. accuracy and reward but not for number of steps due to it getting stuck (Fig. 9). Also, they are only just best and their box and whisker plots are higher (higher mean, quartiles and more outliers) as they often get stuck and crash less frequently. PPO with memory tends to crash and gets stuck infrequently.

6 Safety assurance

The use of the drone navigation recommender system described in this paper in a real-world environment has the potential to cause harm to humans. This harm may either be caused directly by the system (e.g. failures in the system leading to collisions between the drone and other obstacles in the environment), or indirectly result from the system causing a failure to successfully complete a mission (delaying an emergency rescue response for example). If using the drone navigation recommender system in a real-world environment, we must therefore provide confidence that the use of the system will not lead to such harm. We refer to such confidence that behaviour will be safe as "assurance". In this section we will focus on directly caused harm and briefly discuss a strategy for demonstrating assurance in the system and discuss the challenges that would need to be addressed prior to deployment.

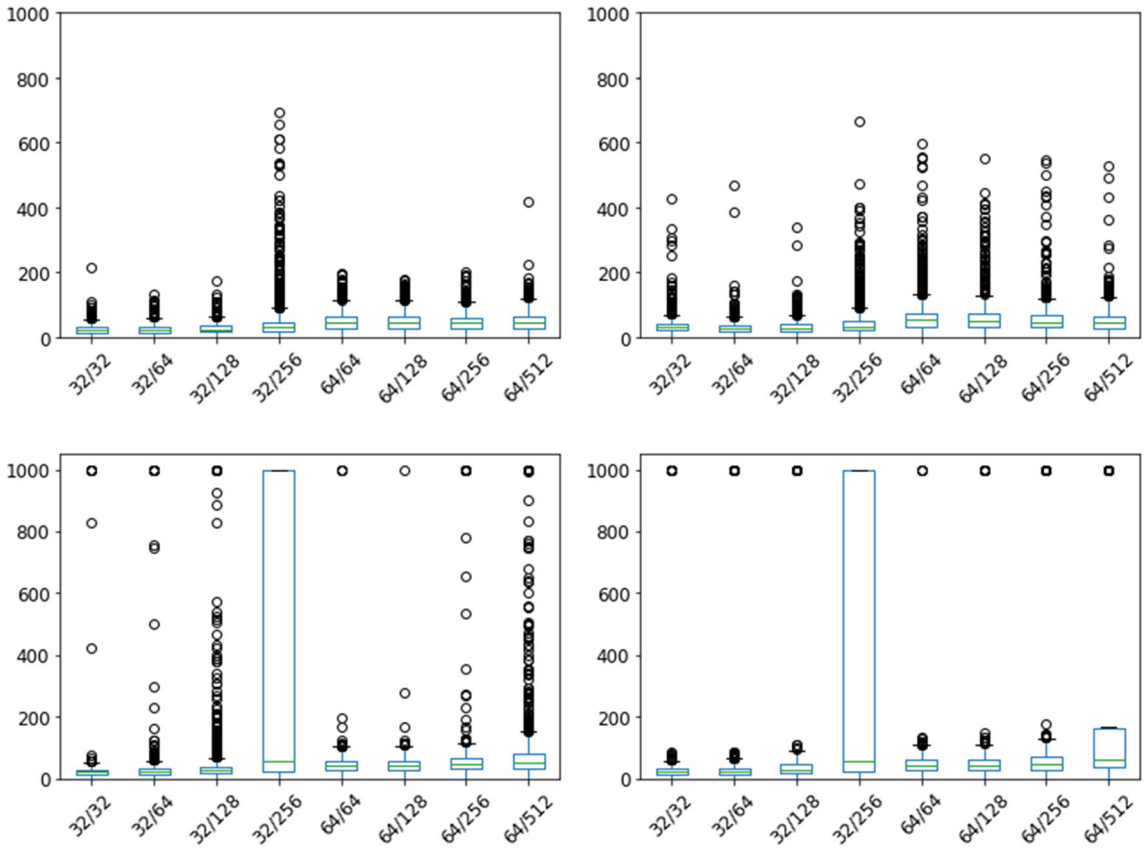

Fig. 9 Box plots of episode length on the *y*-axis (number of steps taken by the drone to find the goal) across 2000 runs with "grid size/number of obstacles" on the *x*-axis for PPO_8 (top left), PPO_{16} (top right),PPO (bottom left) and heuristic (bottom right). Lower values are better (fewer steps taken)

Table 1 Average reward (the average reward achieved by the drone when finding the goal)

Algorithm	Grid size/number of obstacles							
	32/32	32/64	32/128	32/256	64/64	64/128	64/256	64/512
Heuristic	0.92	0.83	0.33	− 1.04	**0.98**	**0.96**	0.86	0.52
PPO	0.94	0.93	0.82	− 0.50	0.97	**0.96**	**0.95**	**0.83**
PPO_8	**0.95**	**0.94**	**0.91**	0.65	0.97	0.95	0.93	**0.83**
PPO_{16}	0.92	0.90	0.89	**0.78**	0.91	0.86	0.78	0.73

The average is taken across 2000 runs and higher values are better (maximum reward is 1.0). The best (highest) value in each column is shown in bold

Table 2 Accuracy (how many times the drone finds the goal)

Algorithm	Grid size/number of obstacles							
	32/32	32/64	32/128	32/256	64/64	64/128	64/256	64/512
Heuristic	1986	1946	1717	1100	**1997**	1981	1873	1504
PPO	1993	1988	1930	1346	1994	**1984**	**1968**	1857
PPO_8	**1994**	**1992**	**1961**	1739	1992	1970	1950	**1858**
PPO_{16}	1985	1961	1944	**1866**	1944	1892	1811	1755

The total is taken across 2000 runs and higher values are better (fewer failures). The best (highest) value in each column is shown in bold

6.1 Identifying safety requirements

In order to define safety requirements for the system we performed a systematic functional failure analysis (FFA) [40]. This analysis considered each function of the system in turn and used a set of standard guidewords as prompts to consider deviations in those functions (function not provided, function provided when not required, function provided incorrectly). The potential worst credible effects of each of those functional deviations were identified, in the form of hazard states of the system that could lead to harm. For those deviations that were determined to be potentially hazardous, potential causes of those deviations by the navigation recommender system were identified. These identified hazard causes can be used to determine a set of safety requirements that must be met by the system.

In Table 3 we extract the results of the FFA for just those deviations that could result in the hazard of collision.

Of these failures, the move function relates to the action of the drone itself, and the avoid collision function relates to the collision avoidance system. We therefore focus here on hazardous failure of the function to determine which way to move, which is implemented by the navigation recommender system. It would be potentially catastrophic for the system to create a plan that would lead to a collision. It is therefore necessary to demonstrate with sufficient confidence prior to putting the system into operation that the system will not produce a plan that results in a collision.

In this case, the results of the FFA are elementary and perhaps quite predictable, but they serve to illustrate how such a technique would contribute to safety assurance. For a more complicated system, FFA is capable of identifying hazardous failures that would not be easily identified through unstructured engineering judgement. Beyond that primary benefit, safety case developers can use its structured nature as grounds for arguing that all hazardous failures have been identified—something that unstructured approaches make difficult. The structure of the FFA also helps to provide traceability from the functional design to the hazards of the system and leads to a tangible artifact that can be reviewed by diverse experts.

Beyond FFA, there are a variety of more complex and specialised techniques with similar properties. Notable examples include FMEA for considering effects of component failures [53], STPA for assessing the overall control structures of a system [30] and ESHA for considering the effects of interactions with a complex environment [14].

6.2 Demonstrating assurance

Demonstrating safety assurance of the navigation recommender system will require the generation of evidence that the defined safety requirement is met. The safety requirement is: ***The navigation recommender system shall never plan a move that leads to collision with an obstacle***.

To provide the necessary confidence that this requirement is satisfied will require assurance in three areas:

- Assurance of the training
- Assurance of the learned model
- Assurance of the overall performance of the drone

The third of these is analogous to system testing. System testing alone, however, is generally deemed to be insufficient for the assurance of safety related systems [18]. For this reason, lower-level verification, analogous to unit level verification of software systems, must also be performed. Similarly, assuring the learned model may not be sufficient unless the training of the model is also assured. In the remainder of this section, we will discuss each of these areas in turn, and consider how assurance could be demonstrated.

6.2.1 Assurance of the training performed

When considering the confidence we can demonstrate in the training that has been carried out for the navigation recommender system, we need to consider both the data used to train the algorithm, and the simulation environment in which the training is performed. From a data perspective, we will consider two critical question for assurance:

1. Are the simulation cases sufficient to ensure robust performance in a real-world environment?

Table 3 Extract of FFA results

Function	Failure mode	Effects
Determine which way to move	(Provided incorrectly) Plans collision	Potentially catastrophic
Move	(Provided when not required) Unplanned move	Potentially catastrophic
	(Provided incorrectly) Move inconsistent with plan	Potentially catastrophic
Avoid collision	(Not provided)	Potentially catastrophic

2. Does the training deal well with the low-probability high-impact edge cases?

The algorithm has been trained in such a way that the safety requirement defined above is met. This training provides evidence to support a safety case for operation. However, this evidence only supports a case for real operation of a drone if the training scenarios provide sufficient examples of real-world scenarios. Although a sufficiently large training set is important to achieve reliable performance of the algorithm, this is not simply a question of quantity of training runs performed. Of equal importance is the "quality" of those training runs with relation to meeting the safety requirement in real-world scenarios. The quality of the training runs will depend upon both their diversity and the inclusion of low-probability edge cases. With regard to diversity, consider, for example, 1000 training runs that present extremely similar scenarios. These are of less value than a single run that exposes the algorithm to a previously unseen scenario.

With regard to low-probability edge cases, it is often unanticipated scenarios that are seen to lead to accidents. Effective training from an assurance perspective must therefore provide as many edge cases as possible. In the training described in this paper, we have been able to achieve a high level of coverage of real-world scenarios, largely due to the abstract nature of the simulation. There are approx. 5×10^{40} combinations of 32 obstacles in a 16×16 grid using C_{32}^{255} (with the goal occupying one cell leaving 255 to place 32 obstacles in). We cannot evaluate them all and would also over-train the neural network preventing it from generalising to new scenarios. Our Unity 3-D simulation uses the C# random number generator to generate the grid layouts. Microsoft describe this number generator as "sufficiently random for practical purposes". In our evaluations, we trained the neural networks for 50 million iterations.

Having evidence of the sufficiency of the training set on its own is insufficient for assurance of the training performed. This provides assurance that the algorithm has been effectively trained to meet the safety requirement in the simulation; it must also be demonstrated that the simulation is sufficiently representative that the learned behaviour in simulation will also be the behaviour observed in the real system. Clearly, the simulation used in this paper for training the navigation recommender system is a very abstract representation of the real-world environment it simulates. From a safety assurance perspective, it is important to understand what simplifying assumptions are made in the simulation, and what impact those assumptions may have on the safety of the system.

In this paper, we have made the following assumptions:

- In the approach here, we assume that all obstacles are equal ($-$ 1 penalty). In reality, some obstacles may be more dangerous than others and we will need to factor this into our model learning in the future, such as using different rewards (penalties) for obstacles.
- There is a single anomaly to be detected or one large anomaly that would override all others and always be detected by the sensors. If there are multiple anomalies, then the sensor data could cause see-sawing of the drone as the highest sensor reading switches between anomaly sites during the drone navigation. However, we would assume that multiple anomalies would require a swarm-based approach so do not consider that here
- In a gas-based anomaly search scenario, if the drone is searching a building for anomalies, then ventilation problems in the building could cause gas to accumulate in particular regions of the building. This could cause false positives. However, these false positives could be eliminated by flying the drone to these sites and circling to assess the accumulation.
- We assume that the drone can find the exact anomaly site once it reaches the "goal" in the simulation. In reality, the drone may need to circle and analyse sensor gradients (differences in sensor reading for adjacent locations) to pinpoint the exact location of the anomaly.
- We have not accounted for defective sensors or erroneous sensor readings. The simulation at this stage assumes that the sensor readings input to the AI are correct and the AI navigates the drone accordingly. A next step for the simulation development is to occasionally perturb the sensor readings, build data cleaning into the anomaly detection algorithms and an accommodation mechanism into the AI so that it can cope.
- We assume that all movement commands are implemented faithfully in the real world (i.e. a command to move North results in the drone moving North in the environment). Under real environmental conditions the movement might be imperfect, so, for example, wind effects may result in a drone being blown off its desired trajectory. If such effects were felt to be important in the target environment, then they could be included in the simulation.
- The AI could lead the drone into a complex cul-de-sac from where it cannot navigate out. Drones have a "return-to-home" mechanism where they follow their flight path back to base. This would be actioned as appropriate. Alternatively, in a collapsed building where the "return-to-home" path is no longer safe than the drone could simply land and await rescue.

6.2.2 Assurance of the learned model

As well as gaining confidence in the safety of the navigation recommender system through the way it has been trained (as discussed in the previous section), it is also important to generate evidence about the sufficiency of the learned model itself. This evidence could be obtained through testing the model in the real world or in the simulator. In this case we have only tested the navigation recommender system in the simulator, and this places natural limits on the level of assurance that can be demonstrated. The confidence that the test evidence provides in the safety of the system will depend upon the following considerations:

1. Are the test cases sufficiently distinct from the training cases?
2. Are the test cases sufficiently representative of real-world situations?

The aim of the testing is to demonstrate that the learned model will satisfy the safety requirements in all real-world scenarios. Although it is not possible to exhaustively test all real-world scenarios, it is important to maximise the coverage of the identified scenarios. In effect we must attempt to simulate raw sensor data in the simulation that corresponds to real-world data as would be sensed by the real sensors in that scenario. As well as coverage of scenarios, a further potential source of uncertainty is the level of correspondence between the approximated raw sensor data in the simulation and the performance of the real sensors. Testing the learned model in this way should provide confidence that the safe behaviour that has been learned by the system from a finite set of training data will also be observed when the system is presented with data upon which it was not trained. If the test cases that are used are too similar to the training cases, then this will not be demonstrated.

The testing described in this paper uses distinct training cases, i.e. 2000 randomly generated grid layouts. Each grid layout is independent of all other grid layouts and the set of layouts should provide good coverage of the scenarios. It is difficult to measure the "quality" of one layout against another when testing. It is the overall coverage of the sequence of layouts that is important rather than each individual layout. Again, it is difficult to measure the quality of one sequence of 2000 layouts against another sequence of 2000 layouts. The C# number generator that we use to randomly generate the training and testing grids is not completely random as it uses a mathematical algorithm to select the numbers, but the numbers are "sufficiently random for practical purposes" according to Microsoft[5]. The current implementation of the C# Random class is based on a modified version of Donald E. Knuth's subtractive random number generator algorithm [27]. This should ensure that a sequence of 2000 layouts provides good coverage during testing.

We also evaluated different grid sizes (different to those used to train the model) and different numbers of obstacles within those grids (again different to those used in training). Each learned model was tested using eight different configurations. This again should make the testing more thorough.

6.2.3 Assurance of the overall performance of the drone

We have dealt here with the assurance of the navigation recommender system. When in operation, this system will be used as one system integrated into a larger drone platform. It is important to provide evidence that the system continues to satisfy its safety requirement when integrated into the drone platform. We have already discussed the role of real-world ("target") testing of the system to gain extra assurance. This not only provides evidence of the system performance in the real-world environment, but also provides evidence that the system performance is not adversely affected by its integration with other components. In particular, target testing would provide the opportunity to identify discrepancies caused by real sensor data. Real-world testing also enables validation of the assumptions described earlier. For instance, the effect of non-simulated environmental factors, such as wind, on the performance of the algorithm could be tested.

The navigation recommender system is just one of many systems that would need to be assured as part of an overall safety assurance case for the drone operation. It is outside the scope of this paper to discuss how a complete assurance case for the drone would be developed. In this discussion, we have, however, provided a strategy by which sufficient assurance could be demonstrated in the navigation recommender system to enable it to be used with confidence as part of a larger drone, or other autonomous platform.

7 Discussion

In this paper, we have demonstrated a drone navigation recommender that uses sensor data to inform the navigation. We adapt the standard PPO approach by incorporating "incremental curriculum learning" (Sect. 3.5). Curriculum learning starts with a simple task and gradually increases the task complexity as it learns. Our incremental curriculum learning dovetails with this by allowing us to vary the length of each lesson in the curriculum until the AI has

[5] https://docs.microsoft.com/en-us/dotnet/api/system.random?view=netframework-4.7.2.

learned that task sufficiently well as demonstrated in Sect. 5. This allows us to gradually learn to navigate complex environments. We added a memory to the AI using a long short-term memory (LSTM) neural network that allows the drone to remember previous steps and prevent it retracing its steps and getting stuck. We train our algorithm using a Unity 3-D simulation environment.

The paper evaluated two configurations of PPO with LSTM against PPO and a simple heuristic technique which functions similar to the PPO (without the learned intelligence). We evaluated PPO with an LSTM memory of length 8 and length 16 to remember where the drone has been. PPO and the heuristic approach form our baseline. The best overall method is PPO with LSTM length 8 which should be used unless the environment is very overcrowded where PPO with LSTM length 16 is best. If the environment is open with very few obstacles then the heuristic is best, e.g. a 64×64 grid with 64 obstacles and PPO is marginally better for 64×64 grids with 128 and 256 obstacles w.r.t. final reward and success rate but takes more steps due to backtracking. The advantage provided by the LSTM memory component to the PPO is that it prevents repetition. Observing the heuristic and PPO approaches, the lack of memory causes the agent to wander side to side when confronted by obstacles, whereas a PPO agent with memory tends to pick a direction and try that way. If it succeeds then it carries on. If it fails, then it backtracks using the memory and tries a different direction. It does not tend to retrace its steps unless it has to as it remembers where it has tried.

The PPO with LSTM approaches tend to wander slightly and deviate from straight lines whereas the heuristic and PPO are more direct. In particular, the LSTM length 16 longer memory rarely gets stuck but takes much longer to train and slightly longer to run as the extra memory increase the degrees of freedom and thus increases the exploration space of the AI so we only recommend it for complex environments.

The advantage provided by the curriculum learning is that it prevents wandering. By starting with a grid with only one obstacle, the AI learns to walk directly to the goal. We then gradually increase the number of obstacles and the AI learns to navigate to the goal with as little wandering as possible. If we do not use curriculum learning and just train from a grid with many obstacles, then the AI learns to wander too much, most noticeably on grids with few obstacles. Hence, using curriculum learning is key to developing a recommender.

While we focussed on drone navigation in hazardous environments in this paper, our navigation model could be deployed in a number of different domains including the detection and identification of chemical leaks, gas leaks, forest fires, disaster monitoring, and search and rescue. It could be used for navigating autonomous ground vehicles (robots) that need to navigate complex and dynamic environments such as Industry 4.0 reconfigurable factories or hospitals to supply equipment; it could be used for delivery robots that deliver parcels or food; and it could be used in agricultural monitoring. The same AI navigator can be used to guide unmanned underwater vehicles used for exploration or maintenance where the environment is ever changing due to currents in the water. It could even be used in video games to navigate characters within the video game. For each of these new domains, the algorithm would remain the same; the only change needed is to select suitable sensors and data to provide the local navigation information required as inputs (i.e. what is immediately to the N, S, E, W and the distance to the target.)

8 Conclusion and future work

Identifying anomalies in environments, buildings and infrastructure is vital to detect problems and to detect them early before they escalate. In this paper, we introduced an anomaly locating drone. It uses a drone-mounted sensor module. We developed an AI-based navigation algorithm that uses the data from these sensors to guide the drone to the exact location of the problem. An anomaly locator is particularly important in safety-critical or hazardous situations for rapid locating and so humans can minimise their exposure to the hazard. The drone may be piloted by a human or fly autonomously.

Whether piloted by a human or an autonomous drone, our navigation algorithm acts as a guide while the pilot focuses on flying the drone safely. Our algorithm is based on the Proximal Policy Optimisation (PPO) deep reinforcement learning algorithm with incremental curriculum learning to improve training and an LSTM recurrent layer to allow the agent to remember where it has been and backtrack when it gets stuck. Deep reinforcement learning algorithms are capable of experience-driven learning for real-world problems making them ideal for our task. We have deliberately configured our algorithm to be generic adaptable and potentially able to work in complex and dynamic environments.

We showed in Sect. 5 that a general algorithm of PPO with LSTM length 8 is best except for very simple environments with very few obstacles where a simple heuristic or PPO with no memory can traverse straight to the problem and very complex environments with many and complex obstacles where PPO with longer short-term memory (LSTM length 16) is best that can retrace its steps further.

Although the human pilot or autonomous drone is responsible for flying the drone while our algorithm acts as

a recommender, it is still important to consider the safety aspects of the system. In Sect. 6, we performed a safety assurance analysis of the system, what safety requirements are needed; and we demonstrated the assurance of training, of the learned model and of the drone.

In the future, we will plug our AI into our wireless sensor module mounted on a drone and navigate our Unity 3-D environment using real sensor data rather than simulated for a qualitative analysis. Following this, we will consider a more rigorous simulation environment such as [47] for a more rigorous quantitative analysis. Next, we will thoroughly qualitatively assess the navigation capabilities by using a human pilot to test the system while flying the drone and using our navigation recommendations. We can then perform the safety assurance analyses recommend in Sect. 6 to ensure safe and trustworthy hardware and software. Once we have completed all steps for the static navigator, we can repeat the process for dynamic environments. This entails training the algorithm to navigate in simulation and then training in the real world followed by qualitative assessments and assurance of safety and trustworthiness.

Funding This work is supported by Innovate UK (Grant 103682) and Digital Creativity Labs jointly funded by EPSRC/AHRC/Innovate UK Grant EP/M023265/1. The work on assurance is funded by the Assuring Autonomy International Programme (www.york.ac.uk/assuring-autonomy).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Appendix

The main Grid-World ML-agents hyper-parameter settings for training. We use the same settings for our PPO, PPO with LTSM length 8 and PPO with LSTM length 16.

- Number of hidden layers: 2
- Number of hidden units in each layer: 64
- Beta—strength of the entropy regularisation, "policy randomness": 2.5e−3
- Gamma—discount factor for future rewards: 0.99
- Lambda—regularisation parameter: 0.95
- Epsilon—acceptable threshold of divergence between the old and new policies: 0.2
- Learning rate—strength of each gradient descent update step: 3.0e−4
- LSTM length: 8 or 16

(See https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-PPO.md for more details of the parameters).

References

1. Abadi M et al (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. http://tensorflow.org/. Software available from tensorflow.org
2. Akyildiz IF, Su W, Sankarasubramaniam Y, Cayirci E (2002) Wireless sensor networks: a survey. Comput Netw 38(4):393–422
3. Anderson K, Gaston KJ (2013) Lightweight unmanned aerial vehicles will revolutionize spatial ecology. Front Ecol Environ 11(3):138–146
4. Aouf A, Boussaid L, Sakly A (2019) Same fuzzy logic controller for two-wheeled mobile robot navigation in strange environments. J. Robot. 2019:2465219
5. Arulkumaran K, Deisenroth MP, Brundage M, Bharath AA (2017) A brief survey of deep reinforcement learning. arXiv preprint arXiv:1708.05866
6. Barnett V, Lewis T (1984) Outliers in statistical data. Wiley series in probability and mathematical statistics: applied probability and statistics. Wiley, Hoboken
7. Beck J, Ciosek K, Devlin S, Tschiatschek S, Zhang C, Hofmann K (2020) Amrl: aggregated memory for reinforcement learning. In: Eighth international conference on learning representations (ICLR). https://www.microsoft.com/en-us/research/publication/amrl-aggregated-memory-for-reinforcement-learning/
8. Bengio Y, Louradour J, Collobert R, Weston J (2009) Curriculum learning. In: Proceedings of the 26th annual international conference on machine learning. ACM, pp 41–48
9. Bristeau PJ, Callou F, Vissiere D, Petit N et al (2011) The navigation and control technology inside the ar. drone micro uav. In: 18th IFAC world congress, Milano, Italy, vol 18, No 1, pp 1477–1484
10. Cao Z, Lin CT (2019) Reinforcement learning from hierarchical critics. arXiv:1902.03079 [cs.LG]
11. Cao Z, Wong K, Bai Q, Lin CT (2020) Hierarchical and non-hierarchical multi-agent interactions based on unity reinforcement learning. In: International conference on autonomous agents and multiagent systems (AAMAS) 2020, demonstration track https://www.youtube.com/watch?v=YQYQwLPXaL4
12. Casbeer DW, Kingston DB, Beard RW, McLain TW (2006) Cooperative forest fire surveillance using a team of small unmanned air vehicles. Int J Syst Sci 37(6):351–360
13. da Silva Assis L, da Silva Soares A, Coelho CJ, Van Baalen J (2016) An evolutionary algorithm for autonomous robot navigation. Procedia Comput Sci 80:2261–2265

14. Dogramadzi S, Giannaccini ME, Harper C, Sobhani M, Woodman R, Choung J (2014) Environmental hazard analysis—a variant of preliminary hazard analysis for autonomous mobile robots. J Intell Robot Syst 76(1):73–117. https://doi.org/10.1007/s10846-013-0020-7
15. Erdelj M, Natalizio E, Chowdhury KR, Akyildiz IF (2017) Help from the sky: leveraging uavs for disaster management. IEEE Pervasive Comput 16(1):24–32
16. Gonzalez L, Montes G, Puig E, Johnson S, Mengersen K, Gaston K (2016) Unmanned aerial vehicles (uavs) and artificial intelligence revolutionizing wildlife monitoring and conservation. Sensors 16(1):97
17. Goodrich MA, Morse BS, Gerhardt D, Cooper JL, Quigley M, Adams JA, Humphrey C (2008) Supporting wilderness search and rescue using a camera-equipped mini uav. J Field Robot 25(1–2):89–110
18. Hawkins R, Kelly T (2009) Software safety assurance—what is sufficient? In: 4th IET international conference on systems safety 2009. Incorporating the SaRS annual conference. IET, pp 1–6
19. Hilder JA, Owens ND, Neal MJ, Hickey PJ, Cairns SN, Kilgour DP, Timmis J, Tyrrell AM (2012) Chemical detection using the receptor density algorithm. IEEE Trans Syst Man Cybern Part C (Appl Rev) 42(6):1730–1741
20. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
21. Hodge V (2011) Outlier and anomaly detection: a survey of outlier and anomaly detection methods. Lambert Academic Publishing, Saarbrücken
22. Hodge V, Austin J (2004) A survey of outlier detection methodologies. Artif Intell Rev 22(2):85–126
23. Hodge VJ, Austin J (2018) An evaluation of classification and outlier detection algorithms. arXiv preprint arXiv:1805.00811
24. Hodge VJ, O'Keefe S, Weeks M, Moulds A (2015) Wireless sensor networks for condition monitoring in the railway industry: a survey. IEEE Trans Intell Transp Syst 16(3):1088–1106
25. Irizarry J, Gheisari M, Walker BN (2012) Usability assessment of drone technology as safety inspection tools. J Inf Technol Constr 17(12):194–212
26. Juliani A, Berges VP, Vckay E, Gao Y, Henry H, Mattar M, Lange D (2018) Unity: a general platform for intelligent agents. arXiv preprint arXiv:1809.02627
27. Knuth DE (1997) The art of computer programming, vol 2, 3rd edn. Seminumerical algorithms. Addison-Wesley, Reading
28. Koh LP, Wich SA (2012) Dawn of drone ecology: low-cost autonomous aerial vehicles for conservation. Trop Conserv Sci 5(2):121–132
29. Kullback S, Leibler RA (1951) On information and sufficiency. Ann Math Stat 22(1):79–86. https://doi.org/10.1214/aoms/1177729694
30. Leveson N, Thomas J (2018) The STPA handbook. MIT. http://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf
31. Levine S, Pastor P, Krizhevsky A, Ibarz J, Quillen D (2018) Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. Int J Robot Res 37(4–5):421–436
32. Li Y, Dai S, Shi Y, Zhao L, Ding M (2019) Navigation simulation of a mecanum wheel mobile robot based on an improved a* algorithm in unity3d. Sensors 19(13):2976
33. Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2015) Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971
34. Matiisen T, Oliver A, Cohen T, Schulman J (2017) Teacher-student curriculum learning. arXiv preprint arXiv:1707.00183
35. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529
36. Neumann PP, Hernandez Bennetts V, Lilienthal AJ, Bartholmai M, Schiller JH (2013) Gas source localization with a micro-drone using bio-inspired and particle filter-based algorithms. Adv Robot 27(9):725–738
37. Ng AY, Coates A, Diel M, Ganapathi V, Schulte J, Tse B, Berger E, Liang E (2006) Autonomous inverted helicopter flight via reinforcement learning. In: Ang MH, Khatib O (eds) Experimental robotics IX. Springer tracts in advanced robotics. Springer, Berlin, pp 363–372
38. Patle B, Ganesh LB, Pandey A, Parhi DR, Jagadeesh A (2019) A review: on path planning strategies for navigation of mobile robot. Def Technol 15(4):582–606. https://doi.org/10.1016/j.dt.2019.04.011
39. Peña JM, Torres-Sánchez J, Serrano-Pérez A, de Castro AI, López-Granados F (2015) Quantifying efficacy and limits of unmanned aerial vehicle (uav) technology for weed seedling detection as affected by sensor resolution. Sensors 15(3):5609–5626
40. Pumfrey DJ (1999) The principled design of computer system safety analyses. Ph.D. thesis, University of York
41. Rashid B, Rehmani MH (2016) Applications of wireless sensor networks for urban areas: a survey. J Netw Comput Appl 60:192–219
42. Rossi M, Brunelli D, Adami A, Lorenzelli L, Menna F, Remondino F (2014) Gas-drone: portable gas sensing system on uavs for gas leakage localization. In: SENSORS, 2014 IEEE. IEEE, pp 1431–1434
43. San Juan V, Santos M, Andújar JM (2018) Intelligent uav map generation and discrete path planning for search and rescue operations. Complexity 2018:6879419
44. Schulman J, Moritz P, Levine S, Jordan M, Abbeel P (2015) High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438
45. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347
46. Singh NH, Thongam K (2019) Neural network-based approaches for mobile robot navigation in static and moving obstacles environments. Intell Serv Robot 12(1):55–67
47. Smyth DL, Glavin FG, Madden MG (2018) Using a game engine to simulate critical incidents and data collection by autonomous drones. arXiv preprint arXiv:1808.10784
48. Sutton RS, Barto AG, Bach F et al (1998) Reinforcement learning: an introduction. MIT Press, Cambridge
49. Tai L, Liu M (2016) Deep-learning in mobile robotics—from perception to control systems: a survey on why and why not. arXiv:1612.07139
50. Tamar A, Wu Y, Thomas G, Levine S, Abbeel P (2016) Value iteration networks. In: Lee DD, Sugiyama M, Luxburg UV, Guyon I, Garnett R (eds) Advances in neural information processing systems, vol 29. Curran Associates, Inc., Red Hook, pp 2154–2162
51. Tomic T, Schmid K, Lutz P, Domel A, Kassecker M, Mair E, Grixa IL, Ruess F, Suppa M, Burschka D (2012) Toward a fully autonomous uav: research platform for indoor and outdoor urban search and rescue. IEEE Robot Autom Mag 19(3):46–56
52. Vanegas F, Gonzalez F (2016) Enabling uav navigation with sensor and environmental uncertainty in cluttered and gps-denied environments. Sensors 16(5):666. https://doi.org/10.3390/s16050666
53. Villemeur A (1992) Reliability, availability, maintainability and safety assessment: volume 1—methods and techniques. Wiley, Chicester

54. Yang J, Liu L, Zhang Q, Liu C (2019) Research on autonomous navigation control of unmanned ship based on unity3d. In: 2019 5th international conference on control, automation and robotics (ICCAR), pp. 422–426. IEEE
55. Zadeh LA (1974) The concept of a linguistic variable and its application to approximate reasoning. In: Fu KS, Tou JT (eds) Learning systems and intelligent robots. Springer, Berlin, pp 1–10
56. Zeng J, Ju R, Qin L, Hu Y, Yin Q, Hu C (2019) Navigation in unknown dynamic environments based on deep reinforcement learning. Sensors 19(18):3837