Aerospace Science and Technology 118 (2021) 107052

Contents lists available at ScienceDirect

Aerospace Science and Technology

www.elsevier.com/locate/aescte

Explainable Deep Reinforcement Learning for UAV autonomous path planning ☆

Lei He [a,*], Nabil Aouf [b], Bifeng Song [a]

[a] School of Aeronautics, Northwestern Polytechnical University, Xi'an, 710072, China
[b] Department of Electrical and Electronic Engineering, City, University of London, London EC1V 0HB, United Kingdom

ARTICLE INFO

Article history:
Received 13 April 2021
Received in revised form 4 July 2021
Accepted 17 August 2021
Available online 25 August 2021
Communicated by Euan Mcgookin

Keywords:
Unmanned Aerial Vehicles (UAVs)
Autonomous navigation
Deep Reinforcement Learning (DRL)
Explainable AI

ABSTRACT

Autonomous navigation in unknown environment is still a hard problem for small Unmanned Aerial Vehicles (UAVs). Recently, some neural network-based methods are proposed to tackle this problem, however, the trained network is opaque, non-intuitive and difficult for people to understand, which limits the real-world application. In this paper, a novel explainable deep neural network-based path planner is proposed for quadrotor to fly autonomously in unknown environment. The navigation problem is modelled as a Markov Decision Process (MDP) and the path planner is trained using Deep Reinforcement Learning (DRL) method in simulation environment. To get better understanding of the trained model, a novel model explanation method is proposed based on the feature attribution. Some easy-to-interpret textual and visual explanations are generated to allow end-users to understand what triggered a particular behaviour. Moreover, some global analyses are provided for experts to evaluate and improve the trained network. Finally, real-world flight tests are conducted to illustrate that our path planner trained in the simulation is robust enough to be applied in the real environment directly.

1. Introduction

In recent years, Unmanned Aerial Vehicles (UAVs) have been widely used in various applications, such as persistent surveillance [1], good delivery [2], remote sensing [3] and wireless networking [4]. To successfully conduct these missions, autonomous navigation and obstacle avoidance are essential capabilities for UAVs to operate intelligently and safety in large unknown complex environments [5].

Generally, there are two main solutions for UAV autonomous navigation and obstacle avoidance. The first one relies on the optimization based on local or global map [6–8]. It is a cascade process, which includes mapping, localization planning and control. This kind of solution can generate nearly optimal trajectories for some optimization objectives such as safety and smoothness. These optimization methods can be also used in the cooperative path planning problem for multiple UAVs [9]. However, this method always requires excessive computation and memory to store the map and to run the optimization algorithms. In addition, these techniques suffer from high drift and noise, impacting the quality of both localization and the map used for planning [10].

Another category is reactive control, which can generate control command from perception information directly [11,12]. This method requires less computation and memory resources because the control signal is obtained using only one forward calculation. Moreover, it does not need to maintain the map during flight. This property gives UAV the capacity to respond to quick changes in the operational environment. This is promising for the real-time implementation on-board micro UAVs with size, weight and power (SWaP) constraints. However, this kind of method is non-optimal because of the lack of global information. Also, the design of this reactive policy relies on the expert experiment.

Observing that the UAV reactive navigation can be treated as a sequential decision-making problem, more and more researchers turn to use learning-based methods. Imanberdiyev et al. [13] developed a high-level control method for autonomous navigation of UAVs using model-based DRL method. He et al. [14] combined bio-inspired monocular vision perception method with a DRL-based local planner to address the micro UAVs navigation problem. They also proposed learning from demonstration method to speed up the training process [15]. Wang et al. [16,17] formulated the problem as a Partially Observable Markov Decision Process (POMDP) and solved it by an online DRL algorithm. These studies

☆ This work was supported by China Scholarship Council No. 201806290175 and Key R & D project of Shaanxi Province under Grant 2020ZDLGY06-05, 2021ZDLGY09-10; City University of London, GB, Postdoc Position.

* Corresponding author.
E-mail addresses: heleidsn@mail.nwpu.edu.cn (L. He), nabil.aouf@city.ac.uk (N. Aouf), bfsong@nwpu.edu.cn (B. Song).

https://doi.org/10.1016/j.ast.2021.107052
1270-9638/

have achieved good results in the simulation environment, but the trained model has not been verified in the real environment. Ross et al. [18] built an imitation learning (IL)-based controller using a small set of human demonstrations and achieved a good performance in real forest environments. However, model trained using IL method cannot outperform the human demonstration. Comparing to the traditional rule-based reactive controller, the control policy trained by DRL can get near optimal actions in the training environment. Also, relying on the powerful feature extraction capacity of Deep Neural Network (DNN), the trained policy can extract feature autonomously without human design. This is expected to provide a better performance.

Except for autonomous navigation problem, DRL method has already been used on UAV, such as morphing control [19] and task allocation [20]. This kind of learning-based method can exceed human performance games [21,22]. Even in some large-scale systems, distributed data-driven intelligent control systems outperform traditional control methods [23]. However, an enormous problem for this kind of learning-based method is that deep learning methods turn out to be "black boxes", which create serious challenges to apply those AI based system in real world. To solve this problem, researchers start focus on the model explanation. This problem falls with the so-called eXplainable AI (XAI) filed [24,25]. There are two kind of methods to increase the transparency of AI models. Using transparency models or using post-hoc XAI techniques. A model is considered to be transparent if by itself it is understandable, such as linear regression, decision trees, rule-based models, etc. This kind of model is usually simple enough to be understood by humans. However, more and more models using deep neural network (DNN) to increase the model prediction accuracy. The DNN model cannot be easily and directly understood by humans. Thus, post-hoc XAI techniques are important to handle such complex models to provide an inner view of those models. Our research group works through the investigation of XAI problem for object classification. Carole et al. invested the model explainability for deep object classification from aerospace vehicles using synthetic aperture radar images [26].

DRL models are usually complex to debug for developers as they rely on many factors, such as environment, reward function, observation and even the algorithms used for training the policy. Thus, there is an urgent demand for explainable DRL (XDRL). Comparing to the burst of XAI research in supervised learning, explainability for RL is hardly explored [27]. Juozapaitis et al. [28] explained the RL agent using reward decomposition. Reward decomposition method is also used in strategic tasks such as StarCraft II [29]. Jung Hoon Lee [30] proposed a method to derive a secondary comprehensible agent from NN-based RL agent and the decisions are made based on simple rules. Beyret et al. [31] proposed an explainable RL for robotic manipulation. Madumal et al. [32] use causal models to derive causal explanations of the behaviour of model-free reinforcement learning agent. A structural causal model is learned during the reinforcement learning phase. The explanations of behaviour are generated based on the counterfactual analysis of the causal model. Although there have been few research works on the explainable RL, no one focuses on the UAV navigation problem. Explainability is critical and essential for the DRL-based UAV navigation system. On the one hand, it's useful for non-expert users to know the reason why the controller choose to turn right rather than turning left when it the UAV faces an obstacle. On the other hand, it supports the network designer to know the network decision making progress to improve its performance.

This paper proposes an explainable DRL method to address the reactive navigation problem for small UAVs with SWaP constraints. The end-to-end navigation deep network is trained in the high-fidelity simulation environment and applied directly to the real world environment. To get better understanding of the trained network, both visual and textual explanations to each model output are provided as local explanations for non-expert users. Moreover, some global explanations are also provided for experts to analyze and improve the deep network transparency.

Although there has been considerable works about UAV navigation using DRL method in indoor environment [33,34], outdoor environment [35] and even in high dynamic environment [36,37], all the proposed trained deep models were evaluated in the simulation environment only. No real experiments have been conducted. In this paper, the controller is trained in the simulation and is applied to the real environment directly. A self-assembled UAV platform is built and some real tests with model explanation are carried out in outdoor environment.

Our main contributions can be summarized as follows:

- A DNN-based reactive controller for UAV path planning is learned using DRL method. The proposed solution can be used for small UAVs with limited computation resources and for indoor/outdoor scenarios requiring rapid reaction to the environment changes.
- A novel explanation framework is proposed to explain the trained DNN-based controller via both visual and texture explanations.
- The DNN model is trained in simulation and evaluated in the real world directly. The real test results show that our network has great potential to adapt from an environment to another and present more computation efficiency comparing to a conventional searching-based approach.
- We provide the for first time an explainable DRL based UAV navigation with real experiments.

2. Preliminaries

2.1. MDP and DRL

In this work, the navigation and obstacle avoidance problem is formulated using MDP. An MDP is defined by a tuple $< S, A, R, P, \gamma >$, which consists of a set of states S, a set of actions A, a reward function $R(s,a)$, a transition function $P(s'|s,a)$, and a discount factor $\gamma \in (0,1)$. In each state $s \in S$, the agent takes an action $a \in A$. After executing the action a in the environment, the agent receives a reward $R(s,a)$ and reaches a new state s', determined from the probability distribution $P(s'|s,a)$.

Solutions for MDPs with finite state and action spaces can be obtained through a variety of methods, such as dynamic programming, especially when the transition probabilities are given. However, in most of the MDPs, the transition probabilities or the reward functions are not available. In this situation, the agent needs to interact with the environment to get some inner information to solve the MDP and this is done by RL method. The goal of RL is to find a policy, π mapping states to actions, that maximizes the expected discounted total reward over the agent's lifetime. This concept is formalized by the action value function: $Q^{\pi}(s,a) = \mathbb{E}^{\pi}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right]$, where $\mathbb{E}^{\pi}$ is the expectation over the distribution of the admissible trajectories $(s_0, a_0, s_1, a_1, \dots)$ obtained the policy π starting from $s_0 = s$ and $a_0 = a$. The action value function can be defined by a tabular mapping of discrete inputs and outputs. However, this tabular mapping is limiting for continuous states or an infinite/large number of states. Different from the traditional RL algorithms, DRL algorithms uses DNN to approximate the action value function, as opposed to tabular functions, to deal with complex problems including infinite/large number of states.

2.2. TD3 algorithm

To get a smooth control command for UAV navigation, an off-policy model-free DRL method, Twin Delayed DDPG (TD3) [38] is adopted for model training. TD3 is the successor of DDPG [39] method. This method addresses the overestimate problem issue of Q-value in DDPG by introducing three critical tricks: clipped double Q-Learning, delayed policy update and target policy smoothing [40].

Different from DDPG algorithm, TD3 concurrently learns two Q-functions, Q_{ϕ_1} and Q_{ϕ_2}, by mean square Bellman error minimization. Moreover, actions used to form the Q-learning target are based on the target policy, $\mu_{\theta_{\text{targ}}}$, but with clipped noise $\epsilon \sim \mathcal{N}(0, \sigma)$ added on each dimension of the action:

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right). \tag{1}$$

This is target policy smoothing which serves as a regularizer for the algorithm. It addresses a particular failure mode that can happen in DDPG: if the Q-function approximator develops an incorrect sharp peak for some actions, the policy will quickly exploit that peak and then have brittle or incorrect behaviour.

Then, TD3 uses clipped double Q-learning. Both Q-functions use a single target, calculated using whichever of the two Q-functions gives a smaller target value:

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{i,\text{targ}}}(s', a'(s')), \tag{2}$$

and then both are learned by regressing to this target:

$$L(\phi_i, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{E}\left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2, \tag{3}$$

where $i = 1, 2$. Lastly, the policy is learned just by maximizing Q_{ϕ_1}:

$$\max_{\theta} \underset{s\sim\mathcal{D}}{\text{E}}\left[Q_{\phi_1}(s, \mu_\theta(s))\right]. \tag{4}$$

Different from DDPG, in TD3, the policy is updated less frequently than Q-functions. This helps to stabilise the training process.

2.3. Feature attribution

Feature attribution is a common method to analyse trained DNN model. Formally, suppose we have a function $F : R^n \to [0, 1]$ that represents a deep neural network and an input $x = (x_1, \ldots, x_n) \in R^n$. An attribution of the prediction at input x relative to a baseline input x' is a vector $A_F(x, x') = (a_1, \ldots, a_n) \in R^n$ where a_i is the contribution of x_i to the prediction $F(x)$. There are two different types of feature attribution algorithms: Shapley-value-based algorithm and gradient-based algorithm. There is a fundamental difference between these two algorithm types.

Shapley value [41] is a classic method to distribute the total gains of a collaborative game to a coalition of cooperating players. It is a fair way to attribute the total gain to the players based on their contribution. Formally, considering a coalitional game, there is a set N (of n players) and a function v that maps subsets of players to the real numbers: $v : 2^N \to R$, with $v(\emptyset) = 0$, where $\emptyset$ denotes the empty set. $v(S)$ is the worth of coalition S, describes the total expected sum of payoffs the members of S can obtain by cooperation. According to the Shapley value, the amount that players i contributed to the game is

$$\phi_i(v) = \sum_{S\subseteq N\setminus\{i\}} \frac{|S|!(n - |S| - 1)!}{n!}(v(S \cup \{i\}) - v(S)) \tag{5}$$

where n is the total number of players and the sum extends over all subsets S of N not containing player i. This equation can be interpreted as follows: for every coalition S without player i, the difference between value function with and without player i is calculated, then, the contribution of player i is equal to the sum of the weighted differences, where $\frac{|S|!(n-|S|-1)!}{n!}$ in equation (5) is the weight. For ML models, we formulate a game for the prediction at each instance. We consider the "total gains" to be the prediction value for that instance, and the "players" to be the model features of that instance. The collaborative game is all of the model features cooperating to form a prediction value. A Shapley-value-based explanation method tries to approximate Shapley values of a given prediction by examining the effect of removing a feature under all possible combinations of presence or absence of the other features.

Besides the Shapley values, gradients can also be used as the feature attribution. A gradient-based explanation method tries to explain a given prediction by using the gradient of the output with respect to the input features. However, the problem with gradients is that they break sensitivity, which is a property that all attribution methods should satisfy. For example, consider a one variable, one ReLU network, $f(x) = 1 - \text{ReLU}(1 - x)$. Suppose the baseline is $x = 0$ and the input is $x = 2$. The output changes from 0 to 1, but the gradient is zero at $x = 2$ because f becomes flat after $x = 1$. Thus, the gradient method gives attribution of 0 to x. This phenomenon has been reported in [42]. To address this problem, Sundararajan et al. [43] proposed Integrated Gradients (IG) algorithm. However, this algorithm requires computing the gradients of the model output on a few different inputs (typically 50) between current feature value and baseline value.

2.4. SHAP and DeepSHAP

According to Section 2.3, Shapley value is a fair way to evaluate the feature attribution of ML model. However, the calculation of Shapley value is computationally expensive because the value function for all the possible feature coalitions has to be calculated according to equation (5). To address this problem, some Shapley value estimation methods are proposed such as Shapley regression values and Shapley sampling values. Lundberg and Lee [44] studied the relationship between these estimation methods and proposed a unified framework for interpreting predictions, named SHAP (SHapley Additive exPlanations). In our case, DeepSHAP, a model-specific Shapley value approximation method, is used to get fast Shapley value estimation.

DeepSHAP [45] is a framework for layer-wise propagation of Shapley values that builds upon DeepLIFT [46]. DeepLIFT can be thought as a fast approximation method of the Shapley values. If model is fully linear, we can get exact Shapley values by summing the attributions along all possible paths between input x_i and the model's output y. However, most networks have non-linear activation function applied after the linear part, such as ReLU, tanh or sigmoid operations. To deal with the non-linear part, DeepLIFT provided both Rescale rule and RevealCancel rule to linearize the non-linear part. For any given reference point, DeepLIFT can get a linearized model near this point using its rules, then back propagate the feature attribution using the linearized model. In our case, because the reference point is fixed, the linearized model only needs to be generated once. After the linear model is generated, Shapley values can be computed efficiently in a single backward pass, which is important for real-time explanation.

2.5. CNN visualization

Understanding the insights of CNN has always been a pain point, though CNN can get excellent predictive performance. In

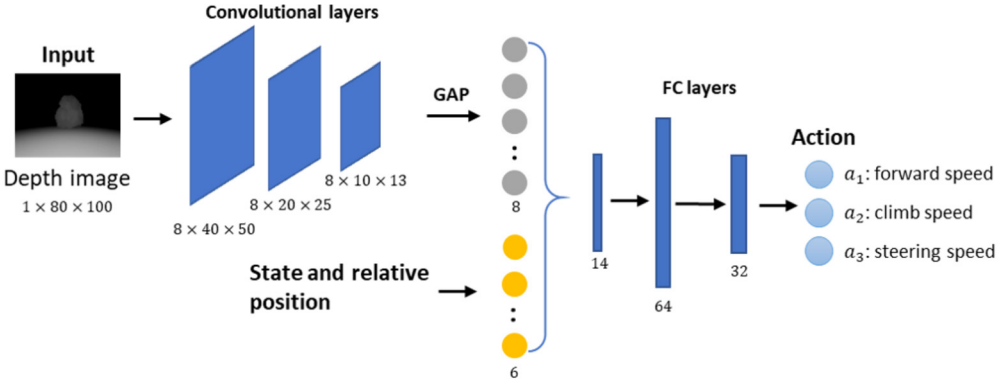


Fig. 1. Network architecture of the controller. The network inputs are raw depth image and UAV states such as current speed and relative position to the goal. Features of the Depth image is extracted using convolutional neural network. Then, global average pooling layer is used to get the intensity of each visual feature and then feed to the fully connected network combined with state features. The outputs are 3 control command includes forward, climb and steering speed.

[47], a deconvolutional network (Deconvnet) approach was proposed to visualize activated pattern in each hidden unit. This method can visualize features individually but is limited as it is hard to summarize all hidden patterns into one pattern. Simonyan et al. [48] visualize partial derivatives of predicted class scores w.r.t. pixel intensities, while Guided Back-propagation [49] makes modifications to 'raw' gradients that result in qualitative improvements. This method can provide fine-grained visualizations.

In [50], the authors proposed Class Activation Map (CAM) using Global Average Pooling (GAP) layer to summarize the activation of the last CNN layer. However, it is only applicable to a particular CNN architecture where the GAP layer is fed directly into the soft-max layer. To address this problem, Grad-CAM [51] method combined feature maps and the gradient signal that does not require any modification in the network architecture. It can be used to off-the-shelf CNN architecture. Grad-CAM uses the gradient information flowing into the last convolutional layer of CNN to assign importance values to each neuron for a particular decision of interest.

3. DRL-based UAV navigation

In this section, a DRL-based reactive controller is proposed to solve the UAV navigation problem in unknown environment. In contrast to conventional simultaneous localization and mapping-based method, the proposed controller navigates the UAV only according to the current sensor data. This kind of controller can make quick response in the complex environment. Also, such reactive controller does not need massive optimization on-board, which is beneficial to the small UAVs with limited computation resources.

3.1. Problem formulation

Reactive navigation in unknown environment is treated as a sequential decision making problem in this paper. At each time step, only the current sensor information is used to generate the control signal. This means that the action a depends only on the current state s. The next state s' depends on the current state s and the action a. This problem can be modelled as a MDP after defining a corresponding reward function $R_a(s, s')$.

Suppose that the UAV takes off from a 3D departure position, denoted as (x_0, y_0, z_0) in the Earth-fixed coordinate frame, and targets at flying to a destination that is denoted as (x_d, y_d, z_d). The observation or state at time t consists of both raw depth image and some UAV state features: $o_t = [o^t_{\text{depth}}, o^t_{\text{state}}]$. The state feature includes the relative position to goal and current velocity information: $o^t_{\text{state}} = [d^t_{xy}, d^t_z, \xi^t, v^t_{xy}, v^t_z, \phi^t]$, where d^t_{xy} and d^t_z denote the distance between the UAV's current position and the destination position in x-y plane and z axis, ξ^t is the relative angle between UAV current first-perspective direction to the destination position, v^t_{xy} and v^t_z are the UAV current speed and ϕ^t is the steering angular speed. Action $a = [v^{\text{cmd}}_{xy}, v^{\text{cmd}}_z, \phi^{\text{cmd}}]$ generated from the policy network $\pi(s)$ consists of 2 linear velocity and 1 angular velocity. These actions are passed to the low-level controller as velocity setpoint command to achieve the goal. The network architecture of the navigation network is shown in Fig. 1.

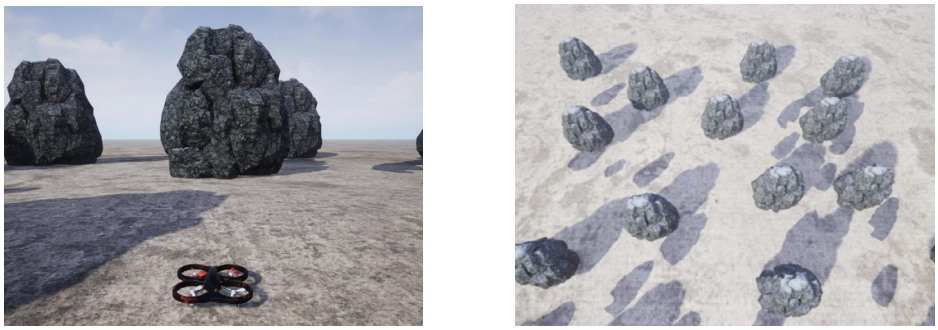
Table 1
Hyperparameters of TD3.

Hyperparameter	Value
mini-batch size	128
replay buffer size	50000
discount factor	0.99
learning rate	0.0003
random exploration steps	2000
square deviation of exploration noise	0.3

3.2. Training environment and setting

The navigation network is trained in AirSim [52] simulator which is built on Unreal Engine. This simulator can provide high fidelity environment with ground truth depth image and a low-level controller to stabilize the UAV. A customized environment is created for training, as shown in Fig. 2. The environment is square with 200 meters on each side. Some stones were randomly placed as obstacles. At the beginning of each episode, the quadrotor takes off from the centre of the environment. The goal position is set randomly on the circle with a radius of 70 meters and centred on the take-off point. The episode terminates when the quadrotor reaches the goal position with an accept radius of 2 meters or crashed on the obstacles. At each time step, the neural network receives the depth image as well as the state information of the quadrotor to generate the velocity setpoint in 3D environment. The controller is running at 10 Hz and the velocity control is realised by the low-level controller provided by AirSim.

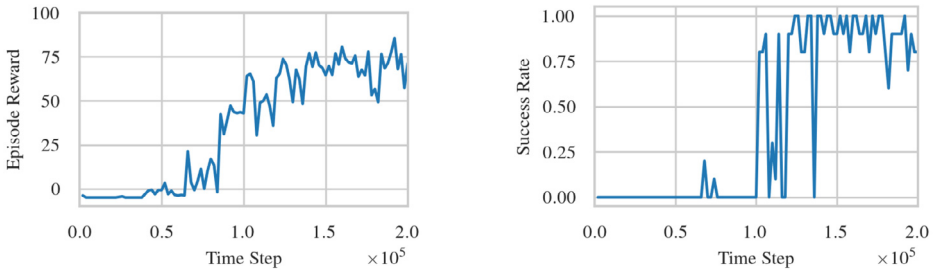
To get a smooth velocity command, we use continuous action space. An off-policy model-free reinforcement learning algorithm, Twin Delayed DDPG (TD3) [38], is adopted for model training. As the successor of the DDPG method, TD3 addresses the overestimate problem issue of Q-value in DDPG by introducing three critical tricks: clipped double Q-Learning, delayed policy update and target policy smoothing [40]. Details about the TD3 algorithm are introduced in Section 2.2. TD3 hyper-parameters are tuned based on massive training. The final hyper-parameters of the algorithm are summarized in Table 1.



(a) The third person perspective from the quadrotor

(b) The top view of the environment

Fig. 2. Customized training environment created using Unreal Engine.



(a) Episode reward

(b) Success rate

Fig. 3. Mean episode reward and success rate versus the training step curves. The success rate is obtained by evaluating each learned policy over 10 randomly generated navigation tasks without action noise. The evaluation is executed every 2k time steps during training.

3.3. Reward function design

Reward function is critical for DRL problem. In general, the reward function for navigation can be simple. For example, only reward for reaching the goal position as soon as possible and punishing for collision is considered. However, because of the huge state space in the navigation task, especially in 3D environment, it is better to introduce continuous reward signal to guide the exploration and speed up the training process. After a lot of testing, a hand-designed reward function is utilized, which consists of a continuous goal approaching reward and some penalty terms:

$$r(s_t) = \begin{cases} 10, & \text{if success} \\ R_{goal} - P_{state}, & \text{otherwise} \end{cases} \tag{6}$$

where $R_{goal} = d(s_{t-1}) - d(s_t)$ is the goal approaching reward and $d(s_t)$ is the *Euclidean distance* from current position to goal position at time t. P_{state} is the penalty term at current step:

$$P_{state} = \omega_1 \cdot C_{obs} - \omega_2 \cdot C_{act} - \omega_3 \cdot C_{pos} \tag{7}$$

where

$$C_{obs} = \frac{d_{safe} - d_{obs}(s_t)}{d_{safe} - d_{min}} \tag{8}$$

is the penalty term to prevent the UAV (quadrotor) from getting close to the obstacle. In equation (8), d_{safe} and d_{min} is the safety distance and minimum distance allowed to the obstacles. $d_{obs}(s_t)$ is the minimum distance to the obstacle at time t. In our training process, $d_{safe} = 5$ and $d_{min} = 1$, which means we give punishment if the quadrotor gets close to the obstacle by 5 meters. When the minimum distance to the obstacle is less than 1 meter, it is considered crashed and this episode terminates. C_{act} and C_{pos} are penalty terms for action, and position error.

3.4. Training result

After defining the reward function, the policy network is trained for 200k time steps (around 1000 episodes) in the simulation environment only. To speed up the training process, the AirSim simulation clock speed is set to 10, which makes the simulator can run 10 times faster than real time. The total training process took about 8 hours on an PC with Intel i7-8700 processor and NVIDIA GeForce GTX1060 GPU. The episode reward and success rate are plotted in Fig. 3. From the training results, the policy gets about 80% success rate when the algorithm converged.

4. Post-hoc explanation method

In this section, we introduce our model explanation method. To keep the network performance, post-hoc explanation approach is used, which means the model is explained after training. Feature attribution is a useful information to generate post-hoc model explanation. As introduced in Section 2.3, in this work, SHAP value is used to measure the feature attribution rather than gradients. Because SHAP value is provably the only distribution with certain desirable properties, which can make better explanation. Specifically, in this paper, the SHAP value is calculated using DeepSHAP method, which is a fast approximate to the Shapley value. DeepSHAP is introduced in Section 2.4.

Different from the traditional image classification task, in our case, the input of the network consists in both depth information (image) and state information (scalar). Hence, our navigation network consists of a Convolutional Neural Network (CNN) perception part to deal with the image information and a Fully Connected Network (FCN) part to fuse the image feature with state feature. Because of this specific kind of network architecture, our expla-

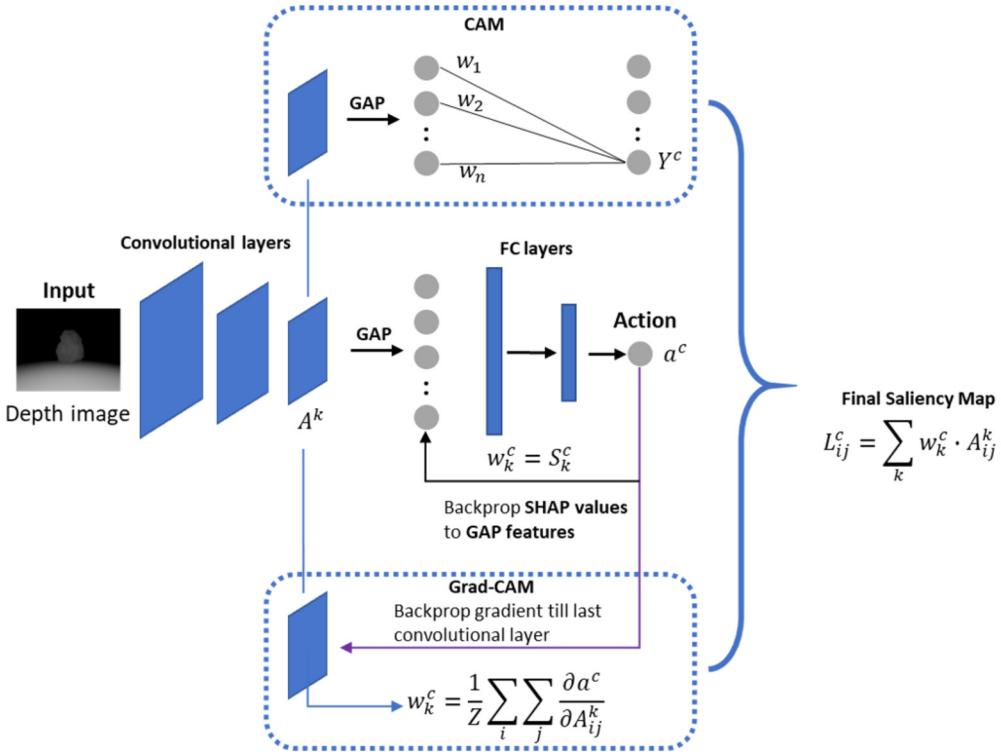


Fig. 4. Our proposed SHAP-CAM method. Different from CAM and Grad-CAM, SHAP values is used to provide better feature attribution of each features rather than gradients.

nation consists of visual explanation part for the image input and text explanation part for the state features.

4.1. Visual explanation

In our problem, the obstacle information is provided by the depth image. A CNN is used to extract the visual feature from the raw depth image. Thus, CNN visualization is important for understanding the output of the learned policy.

To visualize the CNN perception of our network, a new method named SHAP-CAM is proposed, which combining both CAM and SHAP values as introduced in section 2.5. Similar to CAM method, global average pooling (GAP) layer is reserved to summarize the visual feature in the CNN perception network. The output of GAP layer is defined as CNN feature. Different from CAM and Grad-CAM, in our method, the SHAP value is used to determine the importance of the CNN feature rather than gradients. Our proposition is supported by the fact that SHAP value has some unique properties comparing to the gradient, such as efficiency. Finally, a coarse localization map highlighting the important regions in the image is generated by a weighted sum of the last CNN activation map, which is similar to Grad-CAM. The difference between CAM, Grad-CAM and our method is shown in Fig. 4.

4.2. Texture explanation

In addition to the visual explanation, our network also takes some UAV states as input. To get a reasonable explanation of the model output, both image and state input should be considered. To explain the state feature contribution, some texture explanations are provided based on the SHAP values.

Our model has 3 continuous action outputs, horizontal speed v_{xy}^{cmd}, vertical speed v_z^{cmd} and steering angular speed ϕ^{cmd}. To get the textual action description, each action can be divided into 3 parts based on the reference action, as shown in Fig. 5. Assuming that the reference action is the centre value of the action

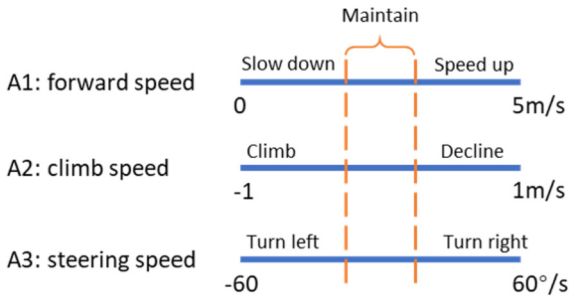


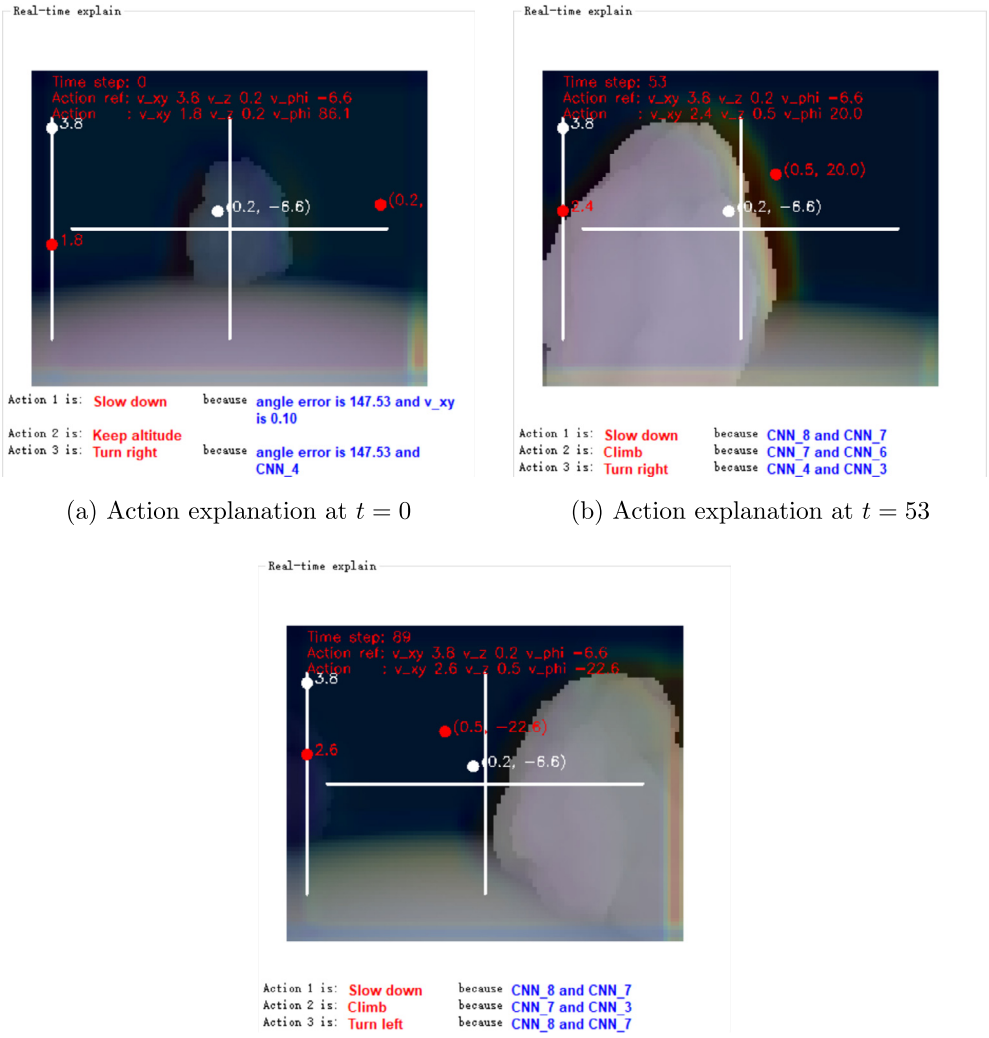
Fig. 5. Action description. Each action is divided into 3 parts. While the prediction fall into the central part, we say it is maintain the current state. Otherwise, there will be a textual description of each action. The final description will be the combination of these three individual descriptions.

space, if the predicted action is similar to the reference action, it is described to maintain current state. If the output action either bigger or smaller than the reference action, a specific text is used to describe the action, such as 'slow down' or 'speed up' for the horizontal speed v_{xy}^{cmd}. The final textual output of the action is the combination of these three action textual descriptions. For example, the action can be described as 'slow down, maintain the altitude and turn right'. As for the explanation, all features are sorted by its attribution to the action output. Then the two most important features are selected to explain the model prediction.

Finally, with both visual and textual explanations, every output of our network can be explained to illustrate the reason of this decision. This kind of information is useful for non-expert to well understand and trust the trained DRL network. Moreover, the explanation only take one forward propagation, which can also provide real-time explanations during flight.

5. Model explanation

In this section, the model trained in section 3 is explained using the explanation method proposed in section 4. The visual explana-



(a) Action explanation at $t = 0$

(b) Action explanation at $t = 53$

(c) Action explanation at $t = 89$

Fig. 6. Action explanation at 3 different time steps. (a) At $t = 0$, the action is slow down, keep altitude and turn right. Mainly because the big angle error to the goal position. (b) At $t = 53$, the action is slow down, climb and turn right, mainly because the image features. From the heat map, we can see the quadrotor is close to the stone and the CNN detected the edge of the stone. (c) At $t = 89$, the action is slow down, climb and turn left. This is also cause of the image feature.

tion part shows the attention of the CNN perception network and the texture explanation part summaries the contribution of other state features. In addition, activation map of the last CNN layer is drawn to show the detailed visual feature extracted by the CNN part. Finally, to help the expert to diagnose and improve the network, some global explanations are also provided to analyse the network based on the evaluation data gathered in 20 continuous episodes.

5.1. Reference input

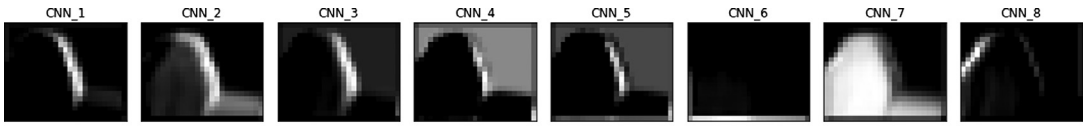
Baselines or references are essential to all explanations [53]. Feature attribution method has to generate the contribution of each feature based on a reference input. Thus, the choice of the reference input is critical for obtaining insightful results [46]. In practice, choosing a good reference would rely on domain-specific knowledge. For instance, in object recognition task, the reference image can be a black image.

In our case, the depth image at the target flight height without any obstacles is chosen as the reference image input. For state input, we set the reference input as $o_{\text{ref}} = [d_{xy} = 70, d_z = 0, \xi = 0, v_{xy} = 0, v_z = 0, \phi = 0]$, which means the UAV only takes off from the start point and has no velocity. The reference image is shown in Fig. 17. Based on this reference input, we can get reference output from the trained network: $v_{xy}^{\text{ref}} = 3.71m/s, v_z^{\text{ref}} = -0.03m/s, \phi^{\text{ref}} = 4.15°/s$.

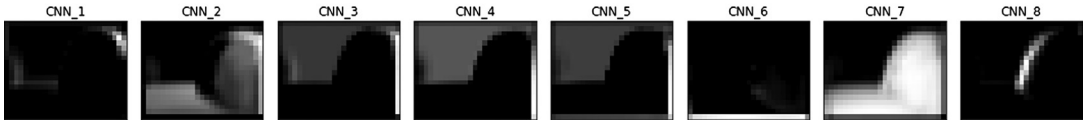
5.2. Local explanation

Local explanation can be generated for every time step. Three specific time steps are chosen to demonstrate the visual and textual explanation in one of the model evaluation episodes. As shown in Fig. 6, at $t = 0$, the action is 'slow down, keep altitude and turn right'. The explanation shows both 'slow down' and 'turn right' action is caused by the 'angular error to goal'. This is because the UAV at $t = 0$ does not face to the goal position, so the UAV need to turn right. At $t = 53$, the action is 'slow down, climb and turn right'. The explanation shows this is mainly caused by the 'CNN feature'. From the heatmap generated using SHAP-CAM, we can see the CNN detected left edge of the stone obstacle. At $t = 89$, the action is 'slow down, climb and turn left'. This is also mainly caused by the 'CNN feature'.

To find out the meaning of the CNN features, the last CNN layer activation maps at both $t = 53$ and $t = 89$ are plotted as shown in Fig. 7. From the activation map, we can see at $t = 53$, that CNN fea-



(a) Last CNN layer activation map at $t = 53$



(b) Last CNN layer activation map at $t = 89$

Fig. 7. Last CNN layer activation map. From this map we can get the meaning of different CNN feature. For example, according to Fig. 6, at $t = 53$, the action 3 is turn right, because CNN_4 and CNN_3 feature. Then, from Fig. 7 (a), CNN_3 and CNN_4 is the right edge of the stone.

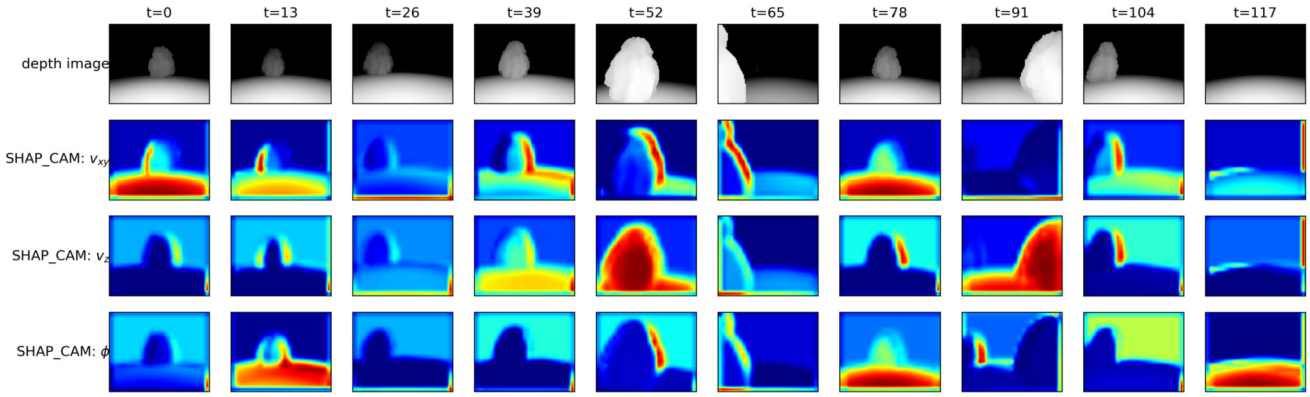


Fig. 8. Depth image and the SHAP-CAM at 10 different time steps. The first line is the input depth image. The second to fourth lines are three SHAP-CAM activation maps for three network outputs separately.

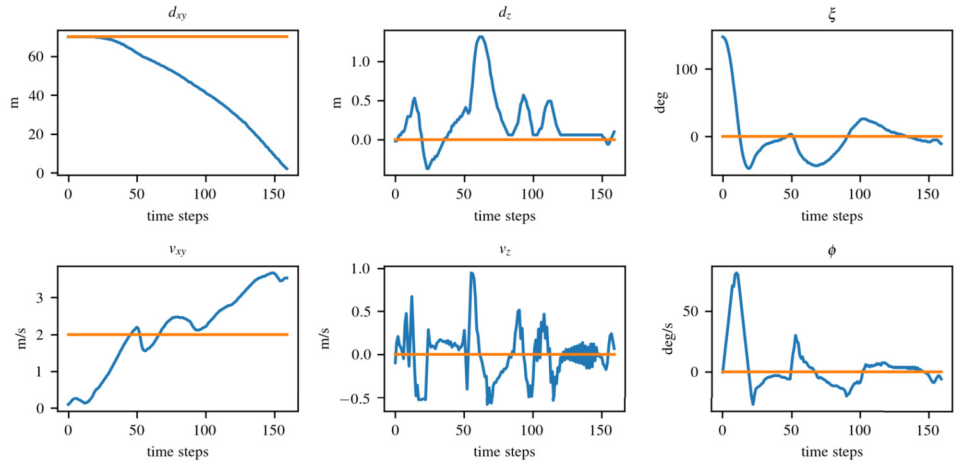


Fig. 9. State features in the evaluation episode. Blue line is the state feature and orange line is the reference state feature value. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

ture 8 is the left and right edges of the obstacle which contributes the most to the slow down action. CNN feature 7 is the obstacle and some ground which contributes to the climb. CNN feature 4 shows the right side edge of the obstacle with some free space background, which leads to the turn right action.

5.3. Global explanation

In addition to the local explanation, some global explanations are provided. First, one episode from the evaluation process is selected and explained. Fig. 8 shows the depth image and the relevant activation map for 3 actions at 10 different time steps. From Fig. 8, at different time step, the network decision-making for different outputs relies on the different visual patterns. Moreover, the control command and state features during the evaluation episode is plotted in Fig. 9 and Fig. 10. From d_{xy} in Fig. 9, the UAV flies towards to the goal position and the distance to goal d_{xy} is reducing over the trajectory. Finally, at $t = 160$, the UAV reached the goal position.

Then, all the feature attributions are summarized over 20 trajectories, 2858 time steps in total. Fig. 11 shows the SHAP summary plot that orders the features based on their importance to the different actions. From the left plot in Fig. 11, the CNN feature

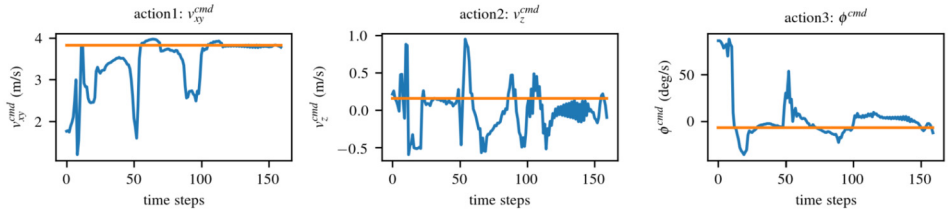


Fig. 10. Network outputs in the evaluation episode. Blue line is the action and orange line is the reference action.

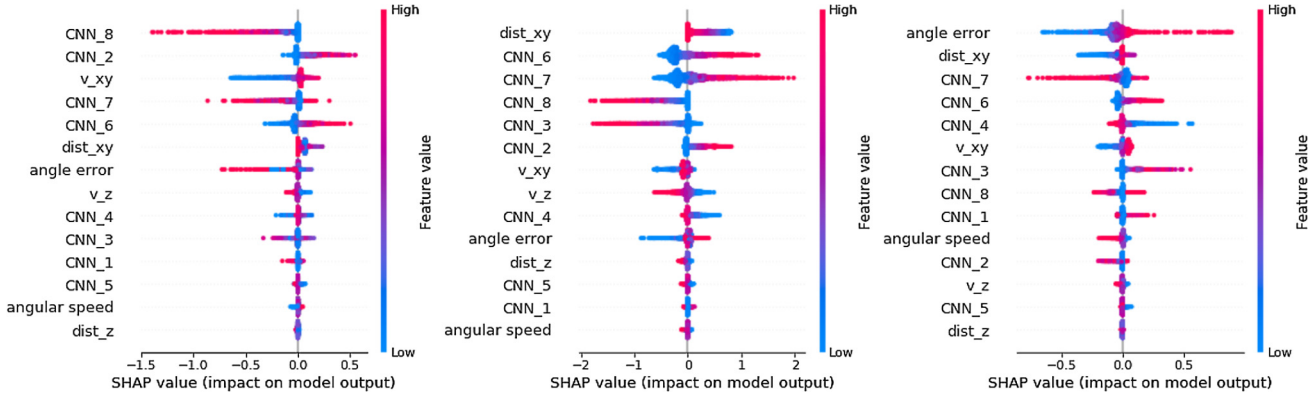


Fig. 11. Feature analysis over 20 trajectories for 3 actions, forward speed v_{xy}^{cmd} (left), vertical speed v_z^{cmd} (middle) and steering speed ϕ^{cmd} (right). For each action, all the features are sorted according to their average attribution. For v_{xy}^{cmd}, the CNN_8 and CNN_2 feature contribute most, then the current forward speed v_xy. For v_z^{cmd}, the distance to goal contributes most. For ϕ^{cmd}, the angle error to goal is the most important feature.

contributes most to action $a1 : v_{xy}^{\text{cmd}}$. Except for the CNN features, the current horizontal velocity v_{xy} and distance to goal d_{xy} are the most important features contributing to $a1 : v_{xy}^{\text{cmd}}$. d_{xy}, v_{xy} and v_z contribute more to $a2 : v_z^{\text{cmd}}$, the vertical velocity command. The angle error ξ is the most important feature contributing to $a_3 : \phi^{\text{cmd}}$.

6. Real world flight test

To validate the performance of our reactive navigation controller, some real world outdoor experiments are carried out. Specifically, to reduce the gap between real and simulation, the DNN model is retrained in the PX4 Simulation In The Loop (SITL) environment, which uses the same flight control stack as the real flight platform.

6.1. Flight platform

A self-assembled quadrotor platform is used to evaluate the trained navigation network, as shown in Fig. 12. The platform is built based on S500 quadrotor framework, equipped with a Pixhawk flight controller. The flight controller can provide low-level attitude and velocity control, as well as the position and velocity information. An Intel RealSense D435i camera is mounted facing forward in front of the quadrotor to get obstacle information. The on-board computer is a NVIDIA Jetson Nano. It is used to run the developed deep neural RL network and generate the velocity control signals. The velocity control signal is sent at 10 Hz to the flight controller as velocity setpoint via serial port.

6.2. Model retraining

Because the flight controller used in AirSim training environment is different from the real platform and to reduce the gap between the simulation and real data, the network is retrained in a custom Gazebo environment. In this Gazebo environment, the controller is running in the PX4 Simulation in the Loop (SITL) configuration, which uses the same flight controller as the real platform [54]. Some trees are placed on the ground as obstacles, as shown in Fig. 13. In addition, to simplify the experiment, the problem

Fig. 12. Self-assembled quadrotor platform used for real flight test.

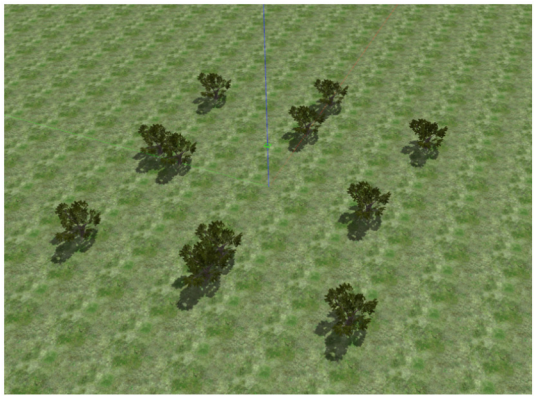
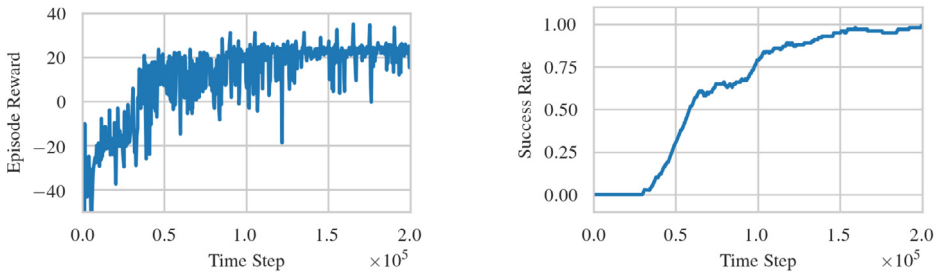
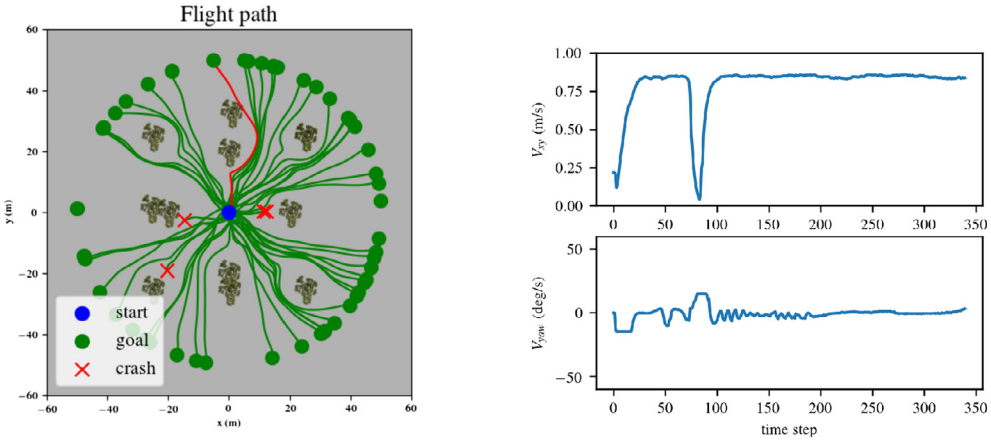


Fig. 13. Gazebo environment for model retraining.



(a) Episode reward (b) Success rate

Fig. 14. Training result in gazebo simulation environment. The model is trained from scratch for 20k time steps.



(a) Flight path for 50 evaluation episodes

(b) Forward speed (up) and steering speed (down) of the selected flight path

Fig. 15. Evaluation result in the training environment.

is limited to a 2D navigation problem as the flight height of the quadrotor is fixed to 5 m. The controller outputs are the setpoints of forward velocity and steering velocity. To keep the quadrotor safe, the maximum forward velocity is limited to 1 m/s. The network is trained in simulation for 20k time steps. Training results are shown in Fig. 14. The success rate is about 95% after training.

6.3. Evaluation in simulation

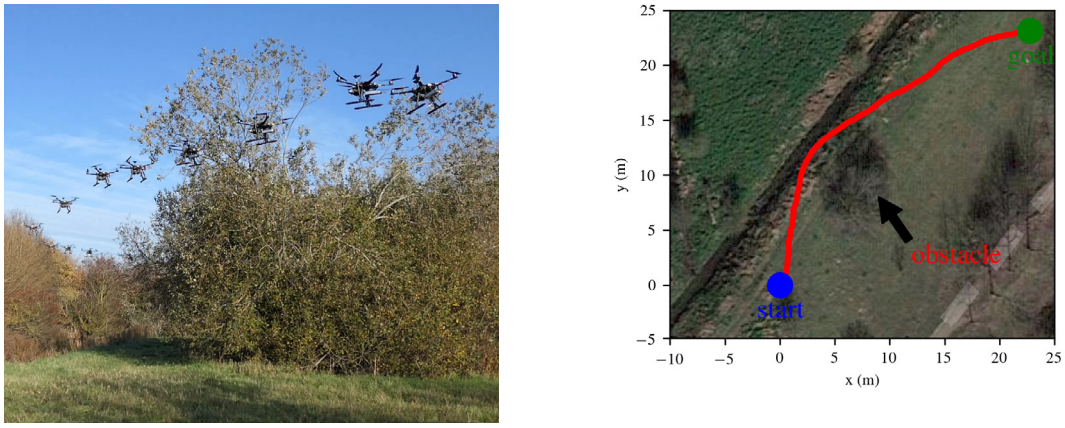
After training, the trained model is evaluated in the training environment firstly. As shown in Fig. 15 (a), the trained model is evaluated for 50 episodes. At the beginning of each episode, the quadrotor takes off from the origin point and flies to the goal position which is distributed on the circle with radius of 50 meters. In 50 evaluation episodes, the UAV reaches goal position for 46 episodes, which is 92% success rate. The crash point is noted by red cross. To demonstrate the state during evaluation, one of the trajectories is selected to do some detailed analysis. The selected trajectory is highlighted in red as shown in Fig. 15a. From the flight path, the quadrotor made an obvious evasive maneuvering to steer away from the obstacle. The forward speed and steering speed of the quadrotor are shown in Fig. 15 (b). When the UAV facing the obstacle at time step 75, our network reduced the speed and turned right to avoid this obstacle.

6.4. Evaluation in the real world

After evaluation in simulation, the trained DRL network is directly deployed to the real flight platform without modification. The real world test environment is shown in Fig. 16a. A large tree is proposed as obstacle. The goal position is set behind the tree and 35 meters away from the start point. Five real tests were conducted at this environment from two different start position and four of them were success. One of the flight paths is shown in Fig. 16b. From the flight path, the trained reactive controller can navigate the quadrotor to avoid the obstacle and reach the goal position finally.

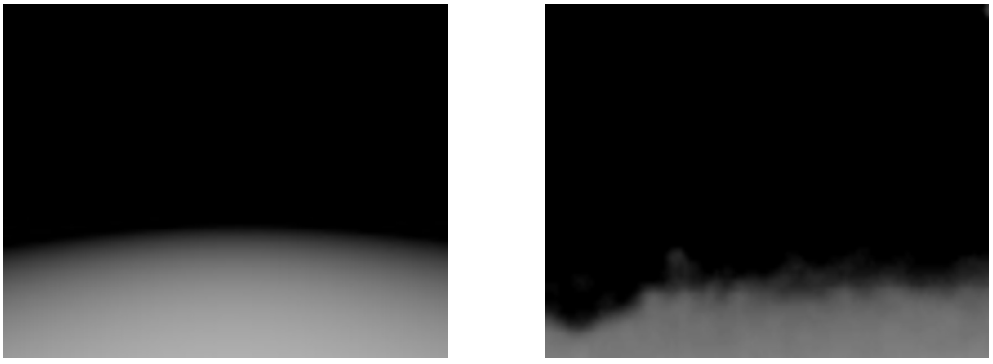
The proposed explanation method is also evaluated in the real environment. The reference image is shown in Fig. 17. Actions predicted at $t = 10$ s, $t = 11$ s and $t = 12$ s are selected separately. As shown in Fig. 19, the forward speed setpoint for all actions is $1m/s$ which is same as the reference input. So, only the steering speed is explained. As shown in Fig. 19 (a), at $t = 10$ s, the network output is 'turn left'. The explanation is that it is mainly because of the 'CNN features'. From the heatmap, we can find that the CNN part detected the edge of the tree. In the next state, at $t = 11$ s, as shown in Fig. 19 (b), the UAV has turned a little bit left from the depth image comparing to $t = 10$ s, the output for steering velocity decrease from -23.3 deg/s to -10.2 deg/s. This is also mainly caused by the 'CNN feature'. At $t = 12$ s, the obstacle totally move out from the field of view as shown in Fig. 19 (c), the actions becomes 'turn right'. The explanation is that this is mainly caused by the 'angle error' to goal position. The network cannot detect the obstacle from the depth image, thus, the network wants to control the quadrotor to face to the goal position.

Fig. 18 shows the forward speed and steering speed during flight. From the experiment, as we expected, some little non-smooth output appears when avoiding the obstacles. This is due to the very challenging problem we are tackling in this paper. The DRL network is trained in the simulation environment only and applied directly to real world conditions. Another reason is the nature of reactive flight with limited camera field of View (FoV). When



(a) Real world test environment (b) Test flight path

Fig. 16. Real world flight test.



(a) Reference depth image in the AirSim simulation (b) Reference depth image in the real world test

Fig. 17. Reference depth image. In our case, the depth image at the target flight height without any obstacles is chosen as the reference image input.

the obstacle disappeared from the FoV, the network will control the quadrotor to face the target direction again. Our developed DRL controller was able to face these challenges and offered a unique solution as the non-smooth output of the network is within tolerance and the quadrotor successfully navigated to the goal position. From the evaluation result shown in Fig. 15b, such non-smooth output also exists in the simulation, but the amplitude is significantly smaller in the simulation than the real world test. Although the real environment is different from the training environment, the trained network still works, which shows our neural network based reactive controller has some ability to be applied in different environments.

In addition, the proposed reactive controller is compared with a traditional obstacle avoidance algorithm. PX4 avoidance project [55] is chosen as the opponent controller. PX4 avoidance project provides ROS nodes for depth sensor fusion and obstacle avoidance, which is based on the 3DVFH+. The local planner of PX4 avoidance can generate waypoint in a vector field histogram including some history information. Flight test result shows that both algorithms can navigate the quadrotor to the final position. However, using the same hardware, the PX4 avoidance system can only run at 10 Hz. In contrast, our deep neural network-based reactive controller can run at 60 Hz. This shows the computational advantage of our reactive controller and it is very important for lightweight UAVs with limited on-board computation resources.

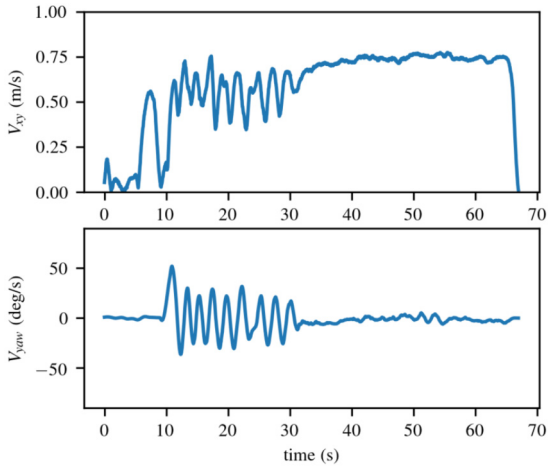
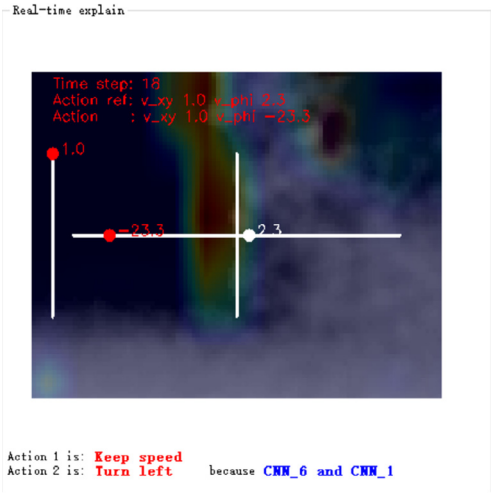


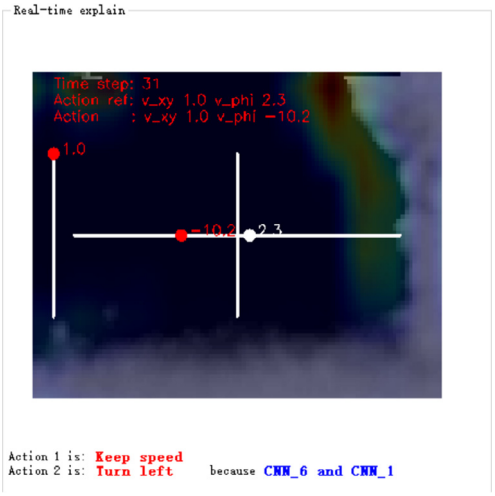
Fig. 18. Forward speed (up) and steering speed (down) during the real flight test.

7. Conclusion

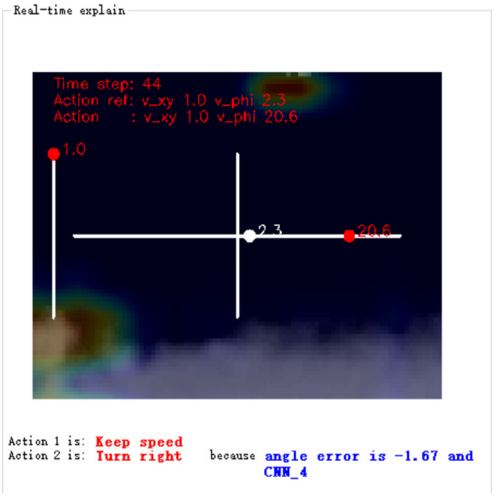
In this paper, the UAV autonomous path planning problem is addressed with DRL technique. Different from other works, the proposed deep network trained in simulation only is evaluated in both simulation and real world environment. Moreover, this paper focused on proposing a new DRL scheme for model explainabil-



(a) Action explanation at $t = 10s$



(b) Action explanation at $t = 11s$



(c) Action explanation at $t = 12s$

Fig. 19. Real world action explanation at 3 different time steps. (a) At $t = 10$ s, the first action is keep speed, because the reference speed and network output for forward speed are both 1 m/s. The steering velocity speed is -23.3 deg/s, which means turn left quickly. This action is generated because of CNN_6 and CNN_1 feature. (b) At $t = 11$ s, the UAV already turned left, the steering output reduced from -23.3 to -10.2 deg/s. (c) At $t = 12$ s, the obstacle went out of the camera view. The action becomes turn right. This is mainly caused of angle error to goal and CNN_4 feature.

ity rather than treating the trained model as a black box. A new saliency map generation method is proposed which combines both CAM and SHAP values. Both visual and textual action explanations can be generated for non-expert users. This is important for the user to get more trust on the application of DRL-based model in the real world environments.

In the future, the model will be fine-trained and improved based on the explanation. Feature attribution method can provide some explanation of the deep neural network, however, it is still pretty shallow. We will look at other methods to make better explanations for the trained network.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Lei He reports financial support was provided by China Scholarship Council No. 201806290175 and Key R & D project of Shaanxi Province under Grant 2020ZDLGY06-05, 2021ZDLGY09-10.

References

[1] Y. Liu, H. Liu, Y. Tian, C. Sun, Reinforcement learning based two-level control framework of UAV swarm for cooperative persistent surveillance in an unknown urban area, Aerosp. Sci. Technol. 98 (2020) 105671.

[2] W.-C. Chiang, Y. Li, J. Shang, T.L. Urban, Impact of drone delivery on sustainability and cost: realizing the UAV potential through vehicle routing optimization, Appl. Energy 242 (2019) 1164–1175.

[3] T.F. Villa, F. Gonzalez, B. Miljievic, Z.D. Ristovski, L. Morawska, An overview of small unmanned aerial vehicles for air quality measurements: present applications and future prospectives, Sensors 16 (7) (2016) 1072.

[4] I. Jawhar, N. Mohamed, J. Al-Jaroodi, D.P. Agrawal, S. Zhang, Communication and networking of UAV-based systems: classification and associated architectures, J. Netw. Comput. Appl. 84 (2017) 93–108.

[5] J. Park, H. Baek, Stereo vision based obstacle collision avoidance for a quadrotor using ellipsoidal bounding box and hierarchical clustering, Aerosp. Sci. Technol. 103 (2020) 105882.

[6] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C.J. Taylor, V. Kumar, Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments, IEEE Robot. Autom. Lett. 2 (3) (2017) 1688–1695.

[7] B. Zhou, F. Gao, L. Wang, C. Liu, S. Shen, Robust and efficient quadrotor trajectory generation for fast autonomous flight, IEEE Robot. Autom. Lett. 4 (4) (2019) 3529–3536.

[8] Y. Song, L.-T. Hsu, Tightly coupled integrated navigation system via factor graph for UAV indoor localization, Aerosp. Sci. Technol. 108 (2021) 106370.
[9] M. Shah, N. Aouf, 3D cooperative Pythagorean hodograph path planning and obstacle avoidance for multiple UAVs, in: 2010 IEEE 9th International Conference on Cybernetic Intelligent Systems, IEEE, 2010, pp. 1–6.
[10] Y. Shin, E. Kim, Hybrid path planning using positioning risk and artificial potential fields, Aerosp. Sci. Technol. (2021) 106640.
[11] S. Paschall, J. Rose, Fast, lightweight autonomy through an unknown cluttered environment: distribution statement: a—approved for public release; distribution unlimited, in: 2017 IEEE Aerospace Conference, IEEE, 2017, pp. 1–8.
[12] H.D. Escobar-Alvarez, N. Johnson, T. Hebble, K. Klingebiel, S.A. Quintero, J. Regenstein, N.A. Browning, R-advance: rapid adaptive prediction for vision-based autonomous navigation, control, and evasion, J. Field Robot. 35 (1) (2018) 91–100.
[13] N. Imanberdiyev, C. Fu, E. Kayacan, I.-M. Chen, Autonomous navigation of UAV by using real-time model-based reinforcement learning, in: 2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), IEEE, 2016, pp. 1–6.
[14] L. He, N. Aouf, J.F. Whidborne, B. Song, Integrated moment-based LGMD and deep reinforcement learning for UAV obstacle avoidance, in: 2020 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2020, pp. 7491–7497.
[15] L. He, N. Aouf, J.F. Whidborne, B. Song, Deep reinforcement learning based local planner for UAV obstacle avoidance using demonstration data, preprint, arXiv: 2008.02521, 2020.
[16] C. Wang, J. Wang, Y. Shen, X. Zhang, Autonomous navigation of UAVs in large-scale complex environments: a deep reinforcement learning approach, IEEE Trans. Veh. Technol. 68 (3) (2019) 2124–2136.
[17] C. Wang, J. Wang, J. Wang, X. Zhang, Deep reinforcement learning-based autonomous UAV navigation with sparse rewards, IEEE Int. Things J. (2020).
[18] S. Ross, N. Melik-Barkhudarov, K.S. Shankar, A. Wendel, D. Dey, J.A. Bagnell, M. Hebert, Learning monocular reactive UAV control in cluttered natural environments, in: 2013 IEEE International Conference on Robotics and Automation, IEEE, 2013, pp. 1765–1772.
[19] D. Xu, Z. Hui, Y. Liu, G. Chen, Morphing control of a new bionic morphing UAV with deep reinforcement learning, Aerosp. Sci. Technol. 92 (2019) 232–243.
[20] X. Zhao, Q. Zong, B. Tian, B. Zhang, M. You, Fast task allocation for heterogeneous unmanned aerial vehicles through reinforcement learning, Aerosp. Sci. Technol. 92 (2019) 588–594.
[21] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[22] M. Moravčík, M. Schmid, N. Burch, V. Lisỳ, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, M. Bowling, Deepstack: expert-level artificial intelligence in heads-up no-limit poker, Science 356 (6337) (2017) 508–513.
[23] M. Kordestani, A.A. Safavi, M. Saif, Recent survey of large-scale systems: architectures, controller strategies, and industrial applications, IEEE Syst. J. (2021).
[24] A.B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, et al., Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI, Inf. Fusion 58 (2020) 82–115.
[25] L. Longo, R. Goebel, F. Lecue, P. Kieseberg, A. Holzinger, Explainable artificial intelligence: concepts, applications, research challenges and visions, in: International Cross-Domain Conference for Machine Learning and Knowledge Extraction, Springer, 2020, pp. 1–16.
[26] C. Belloni, N. Aouf, A. Balleri, J.-M. Le Caillec, T. Merlet, Explainability of deep SAR atr through feature analysis, IEEE Trans. Aerosp. Electron. Syst. (2020).
[27] A. Heuillet, F. Couthouis, N.D. Rodríguez, Explainability in deep reinforcement learning, preprint, arXiv:2008.06693, 2020.
[28] Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, F. Doshi-Velez, Explainable reinforcement learning via reward decomposition, in: IJCAI/ECAI Workshop on Explainable Artificial Intelligence, 2019.
[29] R. Pocius, L. Neal, A. Fern, Strategic tasks for explainable reinforcement learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 10007–10008.
[30] J.H. Lee, Complementary reinforcement learning towards explainable agents, preprint, arXiv:1901.00188, 2019.
[31] B. Beyret, A. Shafti, A.A. Faisal, Dot-to-dot: explainable hierarchical reinforcement learning for robotic manipulation, preprint, arXiv:1904.06703, 2019.
[32] P. Madumal, T. Miller, L. Sonenberg, F. Vetere, Explainable reinforcement learning through a causal lens, preprint, arXiv:1905.10958, 2019.
[33] O. Walker, F. Vanegas, F. Gonzalez, S. Koenig, A deep reinforcement learning framework for UAV navigation in indoor environments, in: 2019 IEEE Aerospace Conference, IEEE, 2019, pp. 1–14.
[34] V.J. Hodge, R. Hawkins, R. Alexander, Deep reinforcement learning for drone navigation using sensor data, Neural Comput. Appl. (2020) 1–19.
[35] B.G. Maciel-Pearson, L. Marchegiani, S. Akcay, A. Atapour-Abarghouei, J. Garforth, T.P. Breckon, Online deep reinforcement learning for autonomous UAV navigation and exploration of outdoor environments, preprint, arXiv:1912.05684, 2019.
[36] G. Tong, N. Jiang, L. Biyue, Z. Xi, W. Ya, D. Wenbo, Uav navigation in high dynamic environments: a deep reinforcement learning approach, Chin. J. Aeronaut. (2020).
[37] C. Yan, X. Xiang, C. Wang, Towards real-time path planning through deep reinforcement learning for a UAV in dynamic environments, J. Intell. Robot. Syst. (2019) 1–13.
[38] S. Fujimoto, H. Van Hoof, D. Meger, Addressing function approximation error in actor-critic methods, preprint, arXiv:1802.09477, 2018.
[39] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, preprint, arXiv: 1509.02971, 2015.
[40] J. Achiam, Spinning up in deep reinforcement learning, 2018.
[41] A.E. Roth, The Shapley Value: Essays in Honor of Lloyd S. Shapley, Cambridge University Press, 1988.
[42] A. Shrikumar, P. Greenside, A. Shcherbina, A. Kundaje, Not just a black box: learning important features through propagating activation differences, preprint, arXiv:1605.01713, 2016.
[43] M. Sundararajan, A. Taly, Q. Yan, Axiomatic attribution for deep networks, preprint, arXiv:1703.01365, 2017.
[44] S.M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in: Advances in Neural Information Processing Systems, 2017, pp. 4765–4774.
[45] H. Chen, S. Lundberg, S.-I. Lee, Explaining models by propagating Shapley values of local components, preprint, arXiv:1911.11888, 2019.
[46] A. Shrikumar, P. Greenside, A. Kundaje, Learning important features through propagating activation differences, preprint, arXiv:1704.02685, 2017.
[47] M.D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in: European Conference on Computer Vision, Springer, 2014, pp. 818–833.
[48] K. Simonyan, A. Vedaldi, A. Zisserman, Deep inside convolutional networks: visualising image classification models and saliency maps, preprint, arXiv:1312.6034, 2013.
[49] J.T. Springenberg, A. Dosovitskiy, T. Brox, M. Riedmiller, Striving for simplicity: the all convolutional net, preprint, arXiv:1412.6806, 2014.
[50] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, A. Torralba, Learning deep features for discriminative localization, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.
[51] R.R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, D. Batra, Grad-cam: visual explanations from deep networks via gradient-based localization, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 618–626.
[52] S. Shah, D. Dey, C. Lovett, A. Kapoor, Airsim: high-fidelity visual and physical simulation for autonomous vehicles, in: Field and Service Robotics, 2017.
[53] D. Kahneman, D.T. Miller, Norm theory: comparing reality to its alternatives, Psychol. Rev. 93 (2) (1986) 136.
[54] Px4-autopilot, https://github.com/PX4/PX4-Autopilot, 2021.
[55] Px4-avoidance, https://github.com/PX4/PX4-Avoidance, 2021.