In: AIAA SciTech Forum 2023, 23-27 January 2023, National Harbor, Maryland, USA
DOI:10.2514/6.2023-0194

Reinforcement Learning for Pan-Tilt-Zoom Camera Control, with Focus on Drone Tracking

Mariusz Wisniewski, Zeeshan A. Rana, Ivan Petrunin
Digital Aviation Research and Technology Centre (DARTeC), Cranfield University, Cranfield, Bedford MK43 0FQ

Reliable detection and tracking of objects using pan-tilt-zoom (PTZ) cameras is an unsolved problem. We attempt to answer whether the use of reinforcement learning (RL) is an appropriate tool for solving it. We present an environment for training RL agents to track a drone using a (PTZ) camera. We also present an agent trained using this environment, which learns to correctly pan, tilt, and zoom the camera to follow a randomly moving drone, using continuous actions. The input into the agent is the RGB image observed by the camera. The agent is rewarded for correctly tracking the drone, and penalized if it loses it from its viewport. We use the recurrent proximal policy optimization (PPO) algorithm with a long short-term memory (LSTM) layer. We find that the agent reliably learns ways of tracking the drone after around 1.4 million steps of training.

I. Introduction

Drones pose a risk to the security of the public infrastructure. Airports are particularly at risk because aircraft operations are safety-critical. In 2018, a drone sighting at Gatwick Airport disrupted air traffic for 3 days. Amateur users of drones have the potential to cause disruptions by negligence and malicious actors may try to use drones to create disruptions on purpose, which may lead to harm. To protect key infrastructure, the British Government laid out a counter-unmanned aircraft strategy [1] which sets out deterring, detecting, and disrupting misuse of drones as a key objective. Through the research presented in this paper, we aim to address the detection of drones.

Common sensor types to detect drones include radar [2–7], radio frequency [8–12], thermal/infrared [13] and optical. In this paper, we focus on the use of optical sensors. The detection of drones can be more precisely broken down into the following:

- Detection: is the drone in the frame recorded by the camera?
- Tracking: if a drone exists in frame one, can we continuously tell its position in frame n?
- Classification: how do we know that the object in the frame is a drone and not another object (e.g. a bird)? Can we tell if the drone is malicious: is it carrying a payload, what is the size of the drone, and what kind of drone model is it?

However, for monitoring large airspace volumes, static cameras are somewhat limited. Field of regard (FOR) is the total area that can be captured by a movable sensor. Field of view (FOV) is the angular cone perceivable by the sensor at a particular time. For a static camera, the FOR and the FOV overlap. A solution to this is to use a matrix of cameras. Another solution is the use of a pan-tilt-zoom (PTZ) camera, as the FOR is much larger than the FOV and depends on its pan/tilt constraints. Using a PTZ camera to monitor an area increases the complexity of the problem, as a control mechanism is required for the camera. The PTZ cameras can be operated by a human (human-in-the-loop control), or be automated by a control system. For optical drone detection, the following methods are commonly used:

- Static camera detection [14–17]: using a single static camera, or a matrix of cameras, where the FOR overlaps with the FOV, to detect a drone in the video feed.
- Human-in-the-loop camera detection [18]: a human is controlling a non-static camera. This can be either a person filming a drone freehand or a human operator controlling a PTZ camera.
- Automated PTZ camera detection [19, 20]: a fully automated control system for controlling the PTZ cameras to detect any flying drones.

There are advantages and disadvantages to each of the methods. Single static camera methods are limited because they can only cover a small area. Most of the videos produced by the drone-vs-bird dataset [17] are filmed using a single static camera. The challenge is to accurately detect the drones, even though they may occupy a very small space in the video feed. Solutions to this usually consist of using some kind of object detection network [21]. For example, Medina et al. [22] present a benchmark that detects drones on the dataset using methods such as Faster-RCNN, SSD512, YOLOv3, and DETR. The problem of the lack of coverage encountered when using a single static camera can be solved

by using a matrix of cameras. A matrix of cameras can be combined with a modification of the background subtraction algorithm to find any moving objects [14]. The downside of this is that the system is quite expensive and generates large amounts of data which need to be processed in real-time, making the system expensive.

Another solution to the coverage problem is using PTZ cameras. These can either be operated by humans or automated. Human-in-the-loop methods do not rely on an algorithm to control the camera. An example of such a system would be handheld videos of drones or security teams with access to camera networks that they can control. There exist some datasets produced by this method. For example, the Anti-UAV dataset [18] contains videos of flying drones, recorded by a PTZ camera operated by a human. In this dataset, we have access to the ground truth (position of the drone in each of the video frames), and the environment changes as the field of view of the camera moves. However, we do not have the possibility of controlling the PTZ of the camera. Hence, such a dataset is good for checking the accuracy of a detection/classification mechanism, but it is impossible to produce and test actions for the camera to take because the videos are already recorded. For optical systems to be fully automated and deployed in real-life, it is important that the control of the camera is considered.

Automated PTZ camera detection relies on a control system to direct the camera to follow the intruding drones. The advantage of this method is that it can observe a much larger space than a single static camera and that it does not have to process a large amount of data in the case of a static camera matrix. It does not rely on human operators and hence should be cheaper to operate. It also has the potential to scale better in the case of a multi-PTZ-camera system. The disadvantage is that there does not exist a huge amount of research on PTZ camera control. Because the FOV of a single PTZ camera is smaller compared to a static camera matrix, its performance might be worse as it is not able to observe the entire area at the same time.

PTZ cameras are commonly used for surveilling an area [23–25]. Rudolph et al. [26] examine using RL to find the most effective strategies for multiple PTZ cameras to surveil an area. Eriksson [27] uses RL to detect targets in a simulated environment. Bisagno et al. [28] use RL to find the optimal methods for crowd surveillance in a simulated environment.

The terms detection, tracking, and classification can be redefined in the context of automated PTZ camera control:

- Detection: does the drone exist in the airspace volume of interest?
- Tracking: if we detect the drone at time t, can we continuously move the pan-tilt-zoom camera to record the drone while it is in the airspace volume of interest?
- Classification: how do we know that the object that the pan-tilt-zoom camera has detected is a drone and not a bird?

Liu et al. [19] show an example of a PTZ control system based on the position of the detected drone in the image. Their architecture consists of detecting flying objects in the video feed, calculating the trajectory, controlling the PTZ, and classifying the objects. Similarly, Svanstrom et al. [20] use a pan-tilt platform to control multiple sensors: a video camera, an infrared camera, a fisheye lens, and a microphone. By fusing the data from the sensors, they are able to correctly classify objects.

One of the problems with research involving PTZ cameras is reproducibility. In order to test the methods presented by other researchers, the same hardware setup is required. A potential way to solve this problem would be to create a synthetic environment to test the ability of algorithms to detect, track, and classify flying drones.

The use of synthetic environments for drone detection and classification has been tried in the literature. [29] use a synthetic environment to generate a dataset to train a neural network to detect, segment, and classify drones. Wisniewski et al. [30] use synthetic images to classify drone models - DJI Phantom, DJI Mavic, and DJI Inspire. The synthetic dataset is used to train a convolutional neural network (CNN) which identifies the drone models in a real-life dataset. Hence, a similar approach could be used to create a synthetic environment which allows for the control of the PTZ camera to detect flying drones. This is actually common in other fields such as robotics, where simulations of robots are used to test algorithms and are later transferred to real life.

The system that we consider in this paper is a reinforcement learning (RL) agent to control the camera actions based on the camera image. The input into the system is an image from the camera, and the output is a PTZ action to control the camera. In the following paragraphs, we explore the literature regarding RL.

RL has been a popular area of research in recent years. Big milestones include AlphaGo beating a human expert at the game GO [31], folding proteins [32], as well as beating humans in video games [33]. Mnih et al. [34, 35] applied deep Q-networks (DQN) to play Atari 2600 games. The input image is input into a CNN, and an action is predicted by the DQN. They found that the trained agent can outperform humans in certain games such as Video Pinball. Chen et al. [36] explored using deep recurrent Q-networks (DQRN) to play the Atari 2600 games. The reason for using a recurrent neural network (RNN) being that it should be able to retain information from older states. Hausknecht et al.

[37] also used DQRNs by adding a long-short-term-memory (LSTM) to the DQN model. They flickered the screens of the Atari games and found that the DQRN model can still perform. They present the use of RNNs as an alternative to standard DQNs by stacking the history of frames in the DQNs input. However, recurrency on its own does not improve the performance of the agent in how well they play the games, and the same performance can be achieved with standard DQNs by stacking the history of observations in the input layer of the CNN.

Lample and Chaplot [38] trained an AI agent that plays the FPS game Doom using only pixels on screen as input to the neural network. They used VizDoom [39] which is a wrapper of the game that allows for reinforcement learning agents to learn to play the game. Further, there exists a number of open-source environments for testing agents such as OpenAI gym [40]. RL environments can be split into discrete action space and continuous action space. In discrete problems, the agent can pick from a number of discrete actions every step. In continuous problems, the agent can select a range of values, usually ranging between -1 and +1 depending on the environment. Mirowski et al. [41] use an asynchronous advantage actor-critic (A3C) algorithm [42] to navigate a maze. The agent is rewarded for reaching the goal, and for finding intermediate rewards. The action space is discrete. They are able to successfully train an agent by using an architecture featuring an LSTM layer, as well as depth predictions.

There are some applications of reinforcement learning to drone control. There exist similarities between the problem of drone control and the problem of PTZ camera control. For example, both of the problems take the image as an input. However, the goal of drone tasks is usually something like obstacle avoidance or reaching a goal by moving the drone. On the other hand, a PTZ camera is stationary and can only rotate and zoom. Hence, we reviewed the literature in this space to see what lessons can be transferred. Munoz et al. [43] use reinforcement learning to train an agent to fly a drone with the aim of reaching a goal. They used two action spaces: a discrete space where the drone can move forward, yaw left, and right, and a continuous space which allows the drone to move freely in the XYZ directions.

Based on the discussed literature, we found there to be a sparse amount of research in the area of using PTZ cameras for tracking drones, and in the area of using RL to track objects. Hence, in this paper, we aim to address these two problems. First, we present a synthetic environment for drone tracking. The environment has been designed with the intention of training an RL agent and is able to interface with a Python client. However, practically, any algorithm can be run to create inputs into the environment and benchmark its performance. Second, we present a working RL agent, trained on this environment, which controls the camera PTZ to correctly track the drone. To the best of the authors' knowledge, RL has not been tried for controlling PTZ cameras to track drones in literature.

This paper is laid out as follows. In section II we present the synthetic environment, the RL agent and the architecture used, and the interface we used to transfer the data between the environment and the agent. In section III we present the results of the training of our RL agent. We also discuss how this compares with the literature. In section IV we conclude our results and lastly, we suggest future work on how to improve this work in section V.

II. Methodology

We attempt to train a reinforcement learning (RL) agent to control a pan-tilt-zoom (PTZ) camera to track a flying drone. To do this, we need an environment, an agent, and an interface between the environment and the agent. The environment contains the PTZ camera and a flying drone. The agent controls how to move the camera. The interface controls the flow of data between the environment and the agent and vice versa. We reward the agent for tracking the drone, and we penalise the agent if it loses sight of the drone.

We created a simple scenario where the drone starts in front of the camera. The drone is assigned a random velocity. In every frame, the drone moves by the given velocity. If the drone goes out of the viewport of the camera, the scenario is terminated, and the agent is penalised. The agent is rewarded for every frame if the drone exists in the camera viewport. However, the reward is only given if the drone covers at least 20 pixels of the image, otherwise, no reward is given. The scenario is terminated after 300 frames.

In the following subsections, we explain the environment, the agent, and the interface in more detail.

A. The Environment

We use Blender, a 3D modelling software package, to create the 3D environment. Although it is not a game engine, it does contains a rendering engine and a Python interface that allows for the programming of parameters. This allows us to create an observation (by rendering the image produced by a camera), change the pan, tilt, and zoom parameters of the camera, and program the 3D model of the drone to move across the 3D space.

The environment needs to implement a reset function and a step function to be compatible with the interface in order to allow for the agent to be trained. Reset resets the environment to the default settings, and the step defines how

the environment changes every step. The step function takes in an action and returns the observation, reward, and termination information.

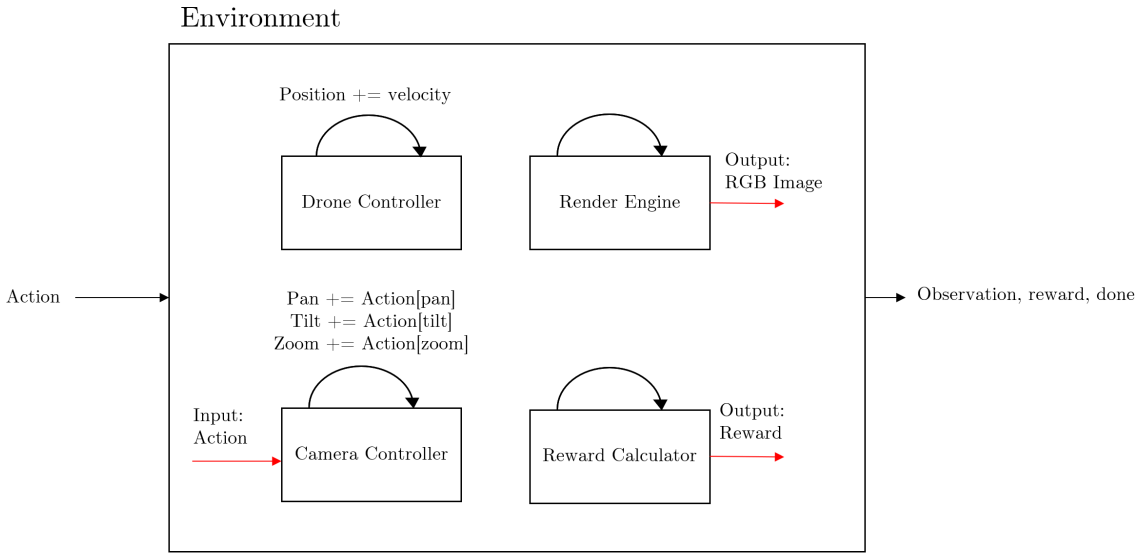

Fig. 1 Environment step. At every step, an action is input into the environment. The camera controller takes the action and updates the camera's pan-tilt-zoom (PTZ) parameters. The drone controller updates the position of the drone. The render engine renders the scene and outputs an RGB image (which is the observation). The reward calculator finds the area occupied by the drone in the image and outputs the reward/termination based on this.

For our PTZ camera, the field of regard (FOR) and the field of view (FOV) do not coincide. We set the limits of the camera movement to be: tilt [30, 330]°, and zoom [20, 200] mm. Yaw is unlimited. These parameters are roughly modelled around the physical camera that we plan to use.

Figure 1 shows the behaviour of the environment during a step. The environment contains 4 main parts: drone controller, camera controller, reward calculator, and render engine. The drone controller controls the movement of the drone. Every time the environment is reset, the camera is moved to the default position, the drone is moved to the position in front of the camera, and the accumulated reward is reset to 0. During each step, the environment takes an action as input. It calculates the reward, renders the image from the camera, and checks whether to terminate the episode. The action is fed into the camera controller and the camera is moved. The drone moves independently from any inputs. Every step, the velocity set at the beginning of the episode is added to its current position. Hence, the path trajectory of the drone is linear. This could be made more complex, but we decided to keep it simple for the scenario. There are boundaries to the drone's position. The boundaries are set to be: $x = [-300, 300]m$, $y = [-300, 300]m$, $z = [-50, 50]m$. If the drone hits the boundary, its velocity is reversed. This is done to prevent the drone from flying too far from the camera.

1. Observation

The image from the camera is rendered at every step. It produces a 160x160x3 pixel RGB image.

2. Action

We use a continuous action space. The possible actions are pan, tilt, and zoom. They have limits set between -1 and +1, and each action is a floating point value. Table 1 shows the continuous actions, their range and how they affect the environment.

Table 1 Continuous action space - action, range and environment response

Action	Range	Environment response
Pan	$[-1, +1]$	+1 translates to +0.02 radians pan to the right
Tilt	$[-1, +1]$	+1 translates to +0.02 radians tilt up
Zoom	$[-1, +1]$	+1 translates to +1 mm of focal length

3. Reward and Termination

The reward is given to the agent for every step that the agent correctly tracks the drone. If the agent loses sight of the drone, it is penalized and the episode is terminated. The reward and termination conditions are based on the area that the drone occupies in the observation frame. After the frame is rendered, the position of the drone vertices is calculated. Using this, a mask of the drone in the image is generated.

(a) Observation RGB frame **(b) Mask**

Fig. 2 RGB observation of the environment, and the mask used to calculate the reward.

Figure 2 shows the observation frame, as rendered by the environment, and a mask which contains a binary representation of the position of the drone. The mask is used to calculate the area of the drone. The area of the drone is then used to calculate the reward.

Table 2 Reward table for different states of the environment

Area occupied by drone (px^2)	State	Reward	Terminate?
Area >= 20	Observing drone	+1	No
20 > Area >= 1	Drone in frame, but is too small	0	No
Area < 1	Drone lost	-100	Yes

Table 2 shows the reward and termination conditions. The agent is rewarded +1 if it keeps the drone in the frame and above 20 px^2 of the observation image. If the drone area falls below this, we no longer add a reward to the agent but we do not terminate the episode. The reasoning for this is that the agent is still looking in the correct direction, but the zoom conditions are not right. If the drone either does not exist in the frame, or it appears too small (less than 1 px^2) it means that the agent has lost sight of the drone. The episode is terminated and -100 reward is added to the agent.

B. The Agent

To train the agent, we use Stable Baselines 3 [44] which provides implementations of common reinforcement learning algorithms such as proximal policy optimization (PPO) [45], advantage actor-critic (A2C) [42], deep-Q-learning (DQN) [34], and others. The library is based on PyTorch [46].

PPO is a policy optimization method that optimizes the objective function using stochastic gradient ascent by batching the observation data and updating the gradients in multiple epochs (as opposed to only updating the gradient once per data sample). The original paper shows that PPO outperforms other policy gradient methods and performs well in environments with a continuous action space.

A2C is a framework which uses asynchronous gradient descent for the optimization of the controller. It performs well during asynchronous training, as all of the parallel actor-learners have a stabilizing effect on training. Unlike DQN, which uses experience replay, A2C uses parallelism by executing multiple agents on multiple instances of the environment.

We chose to use the PPO algorithm to train our agent because it is proven to work well on continuous tasks. Our environment is complex because it involves tracking an object in 3d space, which requires some sort of knowledge of the prior states in the system, for example, to estimate the velocity of the object, and predict where to move the camera. Mnih et al. [34] found that stacking frames worked for DQN. Another solution to this problem is proven to be the use of recurrent networks [36]. We decided to use a recurrent neural network because this is consistent with similar literature [38, 41]. Hence, we use the recurrent PPO algorithm, which uses a long short-term memory (LSTM) layer. For the training, we use default hyperparameters provided by Stable Baselines 3.

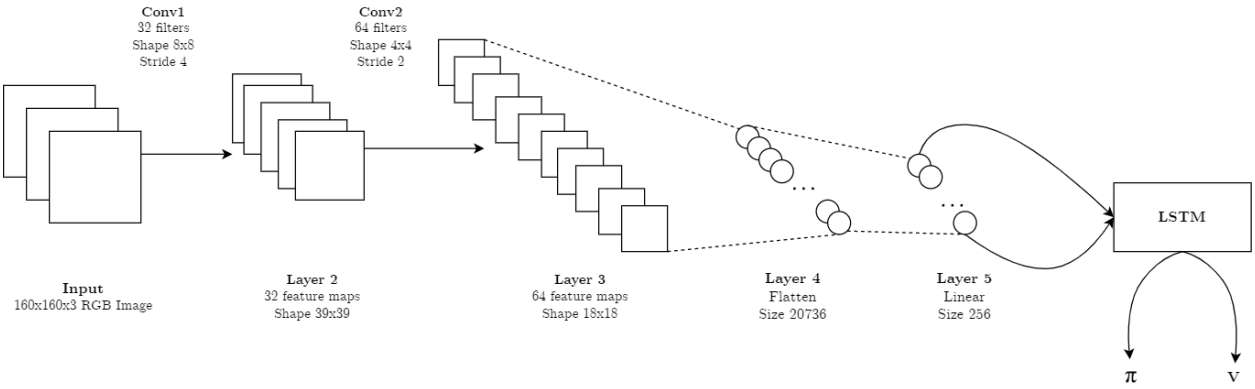

Fig. 3 Architecture of the neural network. The input is an RGB image received from the environment. We then push it through an encoder which consists of two convolutional layers. The features from layer 3 are flattened in layer 4 and outputted to 256 features in layer 5. These features are input into the LSTM layer which predicts the value and the policy.

Figure 3 shows the architecture of our neural network. Because we use an image as an observation, we first use an encoder which uses convolutional layers to extract features. These features are then input into the LSTM layer which predicts the value and the policy.

C. The Interface

An interface is required between the environment and the agent, which transfers data such as the observations, reward, and termination information from the environment to the agent, and the action from the agent to the environment. To achieve this, we use BlendTorch [47] [48], a framework which remotely launches the Blender environment and the PyTorch agent. It interfaces the data between the two using local network transmission. Inside the Blender environment, a remote agent is launched that acts as a middleman between the PyTorch agent and the actual environment. The environment is based on the OpenAI Gym [40] implementation of the environment and includes the reset and step commands.

Figure 4 shows an overview of the communications between the reinforcement learning agent and the Blender environment. The agent (recurrent PPO in our example) communicates with a wrapper of the environment, labelled DroneTrackingEnv. This wrapper then sends the reset and step commands and communicates the action that the agent takes to the remote agent. The remote agent then interfaces with the Blender environment. As the step is taken, a new observation, reward, and termination are generated. This information is sent back to the remote agent which transmits the data to the Python client. The python client also controls the launching and closing of Blender. This allows the launching of multiple environments in parallel.

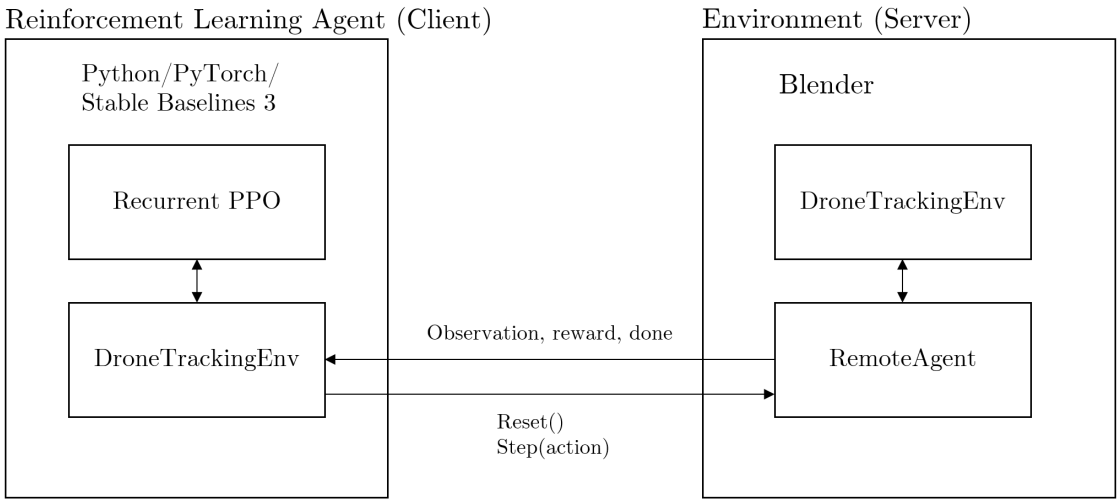

Fig. 4 Data flow between the reinforcement learning agent and the Blender environment

III. Results

We trained the agent using the environment and the settings described in the methodology section.

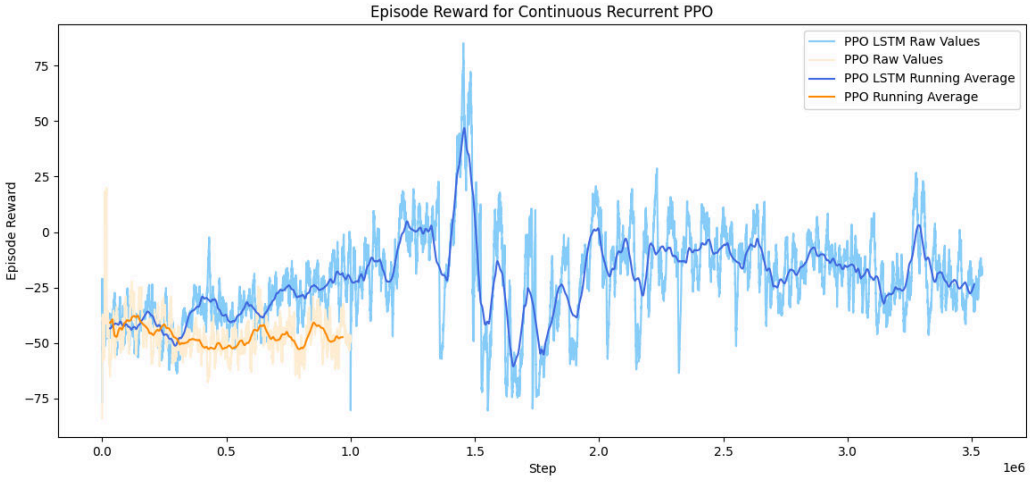

Fig. 5 Episode reward during training

Figure 5 shows the reward during the training with the continuous action space. Two training runs are plotted: proximal policy optimization (PPO) and recurrent PPO (PPO LSTM). Standard PPO training was stopped after a million steps as it was not improving. PPO LSTM showed improvements and the training was carried on until 3.5 million steps. Likewise, Figure 6 shows the consequent episode length. Normal PPO seems unable to learn features about the environment and the episode reward does not increase across the million steps. There is a small spike at the start which is likely to be noise. PPO LSTM slowly increases the reward. Around 1.4 million steps, the episode reward peaks. After that, the reward falls off, although the episode length appears to stay consistent.

The main takeaway is that the PPO LSTM performed better than PPO and, at its best performance, it appears to have been able to learn some actions required to track the drone in the environment. As we did not use observation frame

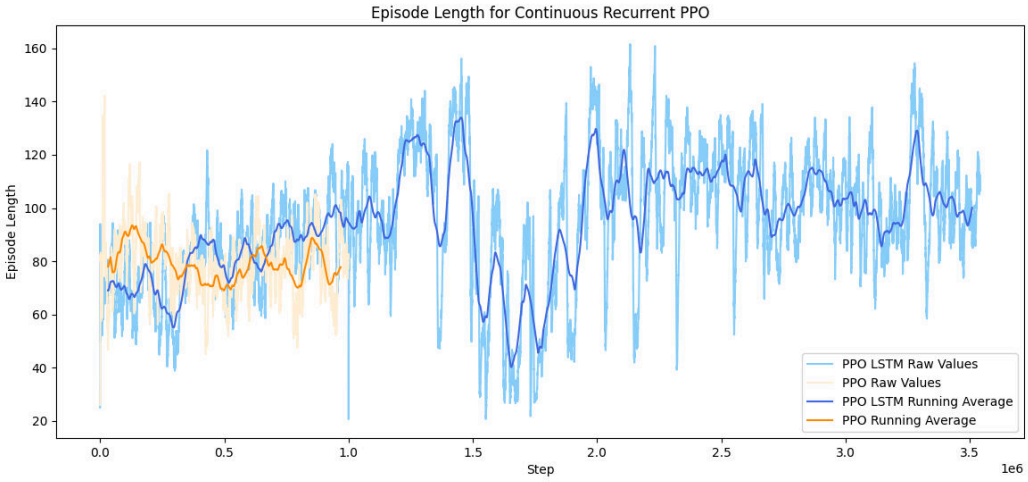

Fig. 6 Episode length during training

(a) Frame 1 **(b) Frame 50** **(c) Frame 100** **(d) Frame 150** **(e) Frame 177**

Fig. 7 Tracking drones through the episode. In this example, the agent correctly pans, tilts, and zooms the camera to keep the drone within the viewport, until frame 177.

stacking for standard PPO, this result appears to be consistent with the literature [36].

Figure 7 shows 5 different frames taken across an episode. The agent used in this episode was the best agent from the training, which appears at around 1.4 million steps in figure 5. In this episode, the agent is correctly able to track the drone for 170 frames which is about 5 seconds in real-time. In frame 177, the drone disappears from the viewport. The interesting thing here is that the drone in frame 150 is actually very far away from the camera (150 meters). Because the agent is zooming in on the drone, it still receives rewards from the environment. After investigating an agent trained at around 3 million steps, we found that the agents did not zoom at all and always kept the zoom setting at the minimum 20 mm. It appears that the agent was trying to optimize for finishing the episodes (to avoid the -100 reward) as opposed to receiving the +1 reward at every step. Zooming in appears to be a risky strategy, as the drone can hit a boundary and quickly reverse the velocity. Perhaps, after another few million episodes of training, the agent would learn some of these more complex behaviours. Hence, it may be the fault of the environment that the agent is not rewarded more for correctly zooming in on the drone to track it.

Lample and Chaplot [38] used a very similar architecture to the one presented in this paper to train an agent to play the video game DOOM. However, they found that the agent did not know the position of the enemies. To solve this, they added a separate hidden layer into the network, with the output corresponding to the features of the game (such as does the enemy exist in the viewport or not). The addition of such a hidden layer (in our context, we could try to predict whether the drone exists in the video frame or not) could potentially improve the performance of the agent. Further, it might improve the performance of the agent on more complex scenarios. This is something that should be examined in the future. Similarly, Mirowski et al. [41] improve the performance by adding depth prediction out of the encoder features in their problem of navigating a maze. In terms of their architecture, they use the asynchronous advantage actor-critic (A3C) algorithm. The input to their network is the observation by the agent, and the output is a discrete

action of how to move around the maze.

IV. Conclusion

Detection and tracking of drones is an important task that needs to be addressed in order to protect key infrastructure from malicious and negligent drone intrusions. Optical drone detection is one method of detecting drones but presents a number of challenges. In this paper, we have identified and addressed the problem of using a pan-tilt-zoom (PTZ) to track drones. To achieve this, we have created a new environment using Blender, and BlendTorch, to train an agent to track a drone by controlling a PTZ camera. We have successfully trained a reinforcement learning (RL) agent using the Proximal Policy Optimization (PPO) algorithm with a long-short-term-memory (LSTM) layer implemented by the Stable Baselines 3 library.

Creating this environment allows us to train RL agents. It also gives us the possibility of running other, non-RL algorithms, and benchmarking the results. We can also compare the results with human-level control, by interfacing the environment with a physical controller.

To the best of the authors' knowledge, this is the first application of RL to a PTZ camera to track a drone. We believe this is an important step in understanding how to better utilize PTZ cameras for the tracking of drones. We have successfully presented a prototype RL agent that has learnt to track a flying drone. This is a novel approach to solving the problem of monitoring large volumes for drones using PTZ cameras. We believe this approach could be generalized to more complex scenarios and environments in the future.

V. Further Work

There are numerous avenues for improving this work. As this is an area of early research, so far we have shown a prototype of the solution inside of a simulation. In the future, a key objective would be to transfer this model to the real world. Sim-to-real transfer is a big problem with reinforcement learning (RL) models [49–52]. An improvement to the environment would be increasing the number of scenarios with varying difficulty, as the current scenario is basic. For example, implement a 'find the drone' scenario where the drone is initialized with random XYZ positions but remains static. An even harder variant would be if the drone was initialized in a random XYZ position but still had a random XYZ velocity. In this scenario, the agent would have to find the drone and then track it. An interesting expansion of this would be to have multi-agent PTZ cameras cooperating together to observe a common space. Ultimately, the performance of Blender is somewhat limited. During the training, we observed around 40 fps for the 160x160x3 images, which seems low. Blender itself is not designed as a game engine, and the rendering is designed for accuracy as opposed to real-time (or better) performance. The lesson learned here is that it was rather complex to set up the interface between the environment and the agent, and the performance was not amazing. Using dedicated game engines such as Unreal Engine or Unity might boost performance in future projects. This is unfortunate because otherwise, Blender offers very good capabilities in terms of programmability and allows quite easily for things like domain randomization and photorealistic images, which may be important for solving the sim-to-real transfer.

Acknowledgments

This project is funded by the Department for Transport courtesy of the Future Aviation Security Solutions (FASS) programme, in collaboration with Aveillant Ltd. and Autonomous Devices Ltd.

References

[1] "UK Counter-Unmanned Aircraft Strategy," , 2019. URL https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/840789/Counter-Unmanned_Aircraft_Strategy_Web_Accessible.pdf.

[2] Jahangir, M., and Baker, C., "Robust Detection of Micro-UAS Drones with L-Band 3-D Holographic Radar," *2016 Sensor Signal Processing for Defence (SSPD)*, IEEE, Edinburgh, United Kingdom, 2016, pp. 1–5. https://doi.org/10.1109/SSPD.2016.7590610.

[3] Dale, H., Baker, C., Antoniou, M., and Jahangir, M., "An Initial Investigation into Using Convolutional Neural Networks for Classification of Drones," *2020 IEEE International Radar Conference (RADAR)*, IEEE, Washington, DC, USA, 2020, pp. 618–623. https://doi.org/10.1109/RADAR42522.2020.9114745.

[4] Fioranelli, F., Ritchie, M., Griffiths, H., and Borrion, H., "Classification of Loaded/Unloaded Micro-drones Using Multistatic Radar," *Electronics Letters*, Vol. 51, No. 22, 2015, pp. 1813–1815. https://doi.org/10.1049/el.2015.3038.

[5] Molchanov, P., Harmanny, R. I., de Wit, J. J., Egiazarian, K., and Astola, J., "Classification of Small UAVs and Birds by Micro-Doppler Signatures," *International Journal of Microwave and Wireless Technologies*, Vol. 6, No. 3-4, 2014, pp. 435–444. https://doi.org/10.1017/S1759078714000282.

[6] Mendis, G. J., Randeny, T., Jin Wei, and Madanayake, A., "Deep Learning Based Doppler Radar for Micro UAS Detection and Classification," *MILCOM 2016 - 2016 IEEE Military Communications Conference*, IEEE, Baltimore, MD, USA, 2016, pp. 924–929. https://doi.org/10.1109/MILCOM.2016.7795448.

[7] Taha, B., and Shoufan, A., "Machine Learning-Based Drone Detection and Classification: State-of-the-Art in Research," *IEEE Access*, Vol. 7, 2019, pp. 138669–138682. https://doi.org/10.1109/ACCESS.2019.2942944.

[8] Yang, S., Qin, H., Liang, X., and Gulliver, T., "An Improved Unauthorized Unmanned Aerial Vehicle Detection Algorithm Using Radiofrequency-Based Statistical Fingerprint Analysis," *Sensors*, Vol. 19, No. 2, 2019, p. 274. https://doi.org/10.3390/s19020274.

[9] Chiper, F.-L., Martian, A., Vladeanu, C., Marghescu, I., Craciunescu, R., and Fratu, O., "Drone Detection and Defense Systems: Survey and a Software-Defined Radio-Based Solution," *Sensors*, Vol. 22, No. 4, 2022, p. 1453. https://doi.org/10.3390/s22041453.

[10] Jamil, S., Fawad, Rahman, M., Ullah, A., Badnava, S., Forsat, M., and Mirjavadi, S. S., "Malicious UAV Detection Using Integrated Audio and Visual Features for Public Safety Applications," *Sensors*, Vol. 20, No. 14, 2020, p. 3923. https://doi.org/10.3390/s20143923.

[11] Al-Sa'd, M. F., Al-Ali, A., Mohamed, A., Khattab, T., and Erbad, A., "RF-based Drone Detection and Identification Using Deep Learning Approaches: An Initiative towards a Large Open Source Drone Database," *Future Generation Computer Systems*, Vol. 100, 2019, pp. 86–97. https://doi.org/10.1016/j.future.2019.05.007.

[12] Nemer, I., Sheltami, T., Ahmad, I., Yasar, A. U.-H., and Abdeen, M. A. R., "RF-Based UAV Detection and Identification Using Hierarchical Learning Approach," *Sensors*, Vol. 21, No. 6, 2021, p. 1947. https://doi.org/10.3390/s21061947.

[13] Zhang, Y., Zhang, Y., Shi, Z., Zhang, J., and Wei, M., "Design and Training of Deep CNN-Based Fast Detector in Infrared SUAV Surveillance System," *IEEE Access*, Vol. 7, 2019, pp. 137365–137377. https://doi.org/10.1109/ACCESS.2019.2941509.

[14] Demir, B., Ergunay, S., Nurlu, G., Popovic, V., Ott, B., Wellig, P., Thiran, J.-P., and Leblebici, Y., "Real-Time High-Resolution Omnidirectional Imaging Platform for Drone Detection and Tracking," *Journal of Real-Time Image Processing*, Vol. 17, No. 5, 2020, pp. 1625–1635. https://doi.org/10.1007/s11554-019-00921-7.

[15] Seidaliyeva, U., Akhmetov, D., Ilipbayeva, L., and Matson, E. T., "Real-Time and Accurate Drone Detection in a Video with a Static Background," *Sensors*, Vol. 20, No. 14, 2020, p. 3856. https://doi.org/10.3390/s20143856.

[16] Mueller, T., and Erdnuess, B., "Robust Drone Detection with Static VIS and SWIR Cameras for Day and Night Counter-UAV," *Counterterrorism, Crime Fighting, Forensics, and Surveillance Technologies III*, edited by H. Bouma, R. J. Stokes, Y. Yitzhaky, and R. Prabhu, SPIE, Strasbourg, France, 2019, p. 10. https://doi.org/10.1117/12.2537940.

[17] Coluccia, A., Fascista, A., Schumann, A., Sommer, L., Ghenescu, M., Avenue, A. O., and Piatrik, T., "Drone-vs-Bird Detection Challenge at IEEE AVSS2019," ????, p. 7.

[18] Jiang, N., Wang, K., Peng, X., Yu, X., Wang, Q., Xing, J., Li, G., Zhao, J., Guo, G., and Han, Z., "Anti-UAV: A Large Multi-Modal Benchmark for UAV Tracking," *arXiv:2101.08466 [cs]*, 2021.

[19] Liu, Y., Liao, L., Wu, H., Qin, J., He, L., Yang, G., Zhang, H., and Zhang, J., "Trajectory and Image-Based Detection and Identification of UAV," *The Visual Computer*, 2020. https://doi.org/10.1007/s00371-020-01937-y.

[20] Svanstrom, F., Englund, C., and Alonso-Fernandez, F., "Real-Time Drone Detection and Tracking With Visible, Thermal and Acoustic Sensors," *arXiv:2007.07396 [cs, eess]*, 2020.

[21] Mediavilla, C., Nans, L., Marez, D., and Parameswaran, S., "Detecting Aerial Objects: Drones, Birds, and Helicopters," *Artificial Intelligence and Machine Learning in Defense Applications III*, edited by J. Dijk, SPIE, Online Only, Spain, 2021, p. 18. https://doi.org/10.1117/12.2600068.

[22] Isaac-Medina, B. K. S., Poyser, M., Organisciak, D., Willcocks, C. G., Breckon, T. P., and Shum, H. P. H., "Unmanned Aerial Vehicle Visual Detection and Tracking Using Deep Neural Networks: A Performance Benchmark," *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, IEEE, Montreal, BC, Canada, 2021, pp. 1223–1232. https://doi.org/10.1109/ICCVW54120.2021.00142.

[23] Davani, S. G., Al-Hadrusi, M. S., and Sarhan, N. J., "An Autonomous System for Efficient Control of PTZ Cameras," *ACM Transactions on Autonomous and Adaptive Systems*, Vol. 16, No. 2, 2021, pp. 1–22. https://doi.org/10.1145/3507658.

[24] Lisanti, G., Masi, I., Pernici, F., and Del Bimbo, A., "Continuous Localization and Mapping of a Pan Tilt Zoom Camera for Wide Area Tracking," , Mar. 2015.

[25] Tang, Y., and Bilodeau, G.-A., "Evaluation of Trackers for Pan-Tilt-Zoom Scenarios," , Nov. 2017.

[26] Rudolph, S., Edenhofer, S., Tomforde, S., and Hähner, J., "Reinforcement Learning for Coverage Optimization Through PTZ Camera Alignment in Highly Dynamic Environments," *Proceedings of the International Conference on Distributed Smart Cameras - ICDSC '14*, ACM Press, Venezia Mestre, Italy, 2014, pp. 1–6. https://doi.org/10.1145/2659021.2659052.

[27] Eriksson, R., "Deep Reinforcement Learning Applied to an Image-Based Sensor Control Task," 2021, p. 53.

[28] Bisagno, N., Xamin, A., De Natale, F., Conci, N., and Rinner, B., "Dynamic Camera Reconfiguration with Reinforcement Learning and Stochastic Methods for Crowd Surveillance," *Sensors*, Vol. 20, No. 17, 2020, p. 4691. https://doi.org/10.3390/s20174691.

[29] Scholes, S., Ruget, A., Mora-Martin, G., Zhu, F., Gyongy, I., and Leach, J., "DronePose: The Identification, Segmentation, and Orientation Detection of Drones via Neural Networks," *arXiv:2112.05488 [cs]*, 2021.

[30] Wisniewski, M., Rana, Z. A., and Petrunin, I., "Drone Model Classification Using Convolutional Neural Network Trained on Synthetic Data," 2022, p. 20.

[31] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D., "Mastering the game of Go with deep neural networks and tree search," *Nature*, Vol. 529, No. 7587, 2016, pp. 484–489. https://doi.org/10.1038/nature16961, URL https://doi.org/10.1038/nature16961.

[32] Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P., and Hassabis, D., "Highly accurate protein structure prediction with AlphaFold," *Nature*, Vol. 596, No. 7873, 2021, pp. 583–589. https://doi.org/10.1038/s41586-021-03819-2, URL https://doi.org/10.1038/s41586-021-03819-2.

[33] OpenAI, "OpenAI Five," https://blog.openai.com/openai-five/, 2018.

[34] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M., "Playing Atari with Deep Reinforcement Learning," , Dec. 2013.

[35] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D., "Human-Level Control through Deep Reinforcement Learning," *Nature*, Vol. 518, No. 7540, 2015, pp. 529–533. https://doi.org/10.1038/nature14236.

[36] Chen, C., Ying, V., and Laird, D., "Deep Q-Learning with Recurrent Neural Networks," 2016, p. 6.

[37] Hausknecht, M., and Stone, P., "Deep Recurrent Q-Learning for Partially Observable MDPs," , Jan. 2017.

[38] Lample, G., and Chaplot, D. S., "Playing FPS Games with Deep Reinforcement Learning," , Jan. 2018.

[39] Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W., "ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning," , Sep. 2016.

[40] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W., "OpenAI Gym," , 2016.

[41] Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A. J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., Kumaran, D., and Hadsell, R., "Learning to Navigate in Complex Environments," , Jan. 2017.

[42] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K., "Asynchronous Methods for Deep Reinforcement Learning," , Jun. 2016.

[43] Muñoz, G., Barrado, C., Çetin, E., and Salami, E., "Deep Reinforcement Learning for Drone Delivery," *Drones*, Vol. 3, No. 3, 2019, p. 72. https://doi.org/10.3390/drones3030072.

[44] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N., "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, Vol. 22, No. 268, 2021, pp. 1–8. URL http://jmlr.org/papers/v22/20-1364.html.

[45] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal Policy Optimization Algorithms," , Aug. 2017.

[46] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," 2019, p. 12.

[47] Heindl, C., Brunner, L., Zambal, S., and Scharinger, J., "BlendTorch: A Real-Time, Adaptive Domain Randomization Library," , Oct. 2020.

[48] Heindl, C., Zambal, S., and Scharinger, J., "Learning to Predict Robot Keypoints Using Artificially Generated Images," , Jul. 2019.

[49] Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V., and Hutter, M., "Learning Agile and Dynamic Motor Skills for Legged Robots," *Science Robotics*, Vol. 4, No. 26, 2019, p. eaau5872. https://doi.org/10.1126/scirobotics.aau5872.

[50] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P., "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Vancouver, BC, 2017, pp. 23–30. https://doi.org/10.1109/IROS.2017.8202133.

[51] James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., and Bousmalis, K., "Sim-to-Real via Sim-to-Sim: Data-efficient Robotic Grasping via Randomized-to-Canonical Adaptation Networks," *arXiv:1812.07252 [cs]*, 2019.

[52] Tremblay, J., Prakash, A., Acuna, D., Brophy, M., Jampani, V., Anil, C., To, T., Cameracci, E., Boochoon, S., and Birchfield, S., "Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization," *arXiv:1804.06516 [cs]*, 2018.

Reinforcement learning for pan-tilt-zoom camera control, with focus on drone tracking

Wisniewski, Mariusz

2023-01-19

Wisniewski M, Rana ZA, Petrunin I. (2023) Reinforcement learning for pan-tilt-zoom camera control, with focus on drone tracking. In: AIAA SciTech Forum 2023, 23-27 January 2023, National Harbor, Maryland, USA. Paper number AIAA 2023-0194

https://doi.org/10.2514/6.2023-0194