Leveraging Deep Reinforcement Learning For Active Shooting Under Open-World Setting

A. Tzimas, N. Passalis, A. Tefas

LEVERAGING DEEP REINFORCEMENT LEARNING FOR ACTIVE SHOOTING UNDER OPEN-WORLD SETTING

A. Tzimas, N. Passalis, and A. Tefas

Department of Informatics, Aristotle University of Thessaloniki, Greece
E-mail:{tzimasak, passalis, tefas}@csd.auth.gr

ABSTRACT

Recent advances in Deep Reinforcement Learning (DRL) led to the development of powerful agents that can learn how to perform complicated tasks in an end-to-end fashion operating directly on raw unstructured data, e.g., images. However, the real world performance of such methods critically relies on the quality of the simulation environments used for training them. The main contribution of this paper is the development of a realistic simulation environment, by employing a state-of-the-art graphics engine, for training DRL agents that are able to control a drone for performing active shooting. In contrast with previous approaches, that solely relied on simplistic constrained datasets, the environment employed in this work supports a challenging open-world setting, providing a solid step towards developing effective RL methods for various drone control tasks. An appropriate reward shaping approach is also introduced in this work, ensuring that the agent will behave as expected, avoiding erratic movements, as demonstrated through the conducted experiments.

Index Terms— Active Shooting, Deep Reinforcement Learning, Open-World Setting

1. INTRODUCTION

Deep Learning (DL) provided powerful information analysis tools that allowed for developing a wide range of intelligent control methods [1, 2, 3, 4]. These applications range from developing autonomous cars [2, 3], to drones that can autonomously perform various challenging tasks, such following trails in forests [4], assisting rescue operations [5], and covering various public events [6]. Early approaches, e.g., [4], relied on DL for performing the visual information analysis, the output of which was used by traditional control approaches,e.g., proportional-integral-derivative (PID) controllers [7, 8], to perform the task at hand. These approaches managed to solve a wide range of difficult problems that required performing information analysis from highly unstructured data, e.g., detecting and/or tracking the objects of interest in an image [9, 10], by employing powerful DL-based analysis methods.

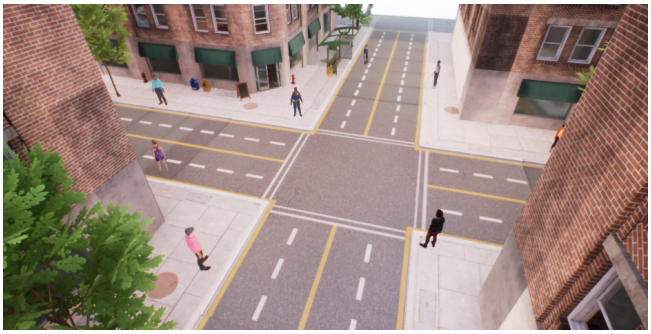


Fig. 1: Open-world simulation environment developed in this paper

However, these approaches suffer from a significant drawback: information analysis is performed independently from control. That is, we typically first design a DL method for extracting some handcrafted high-level features, e.g., bounding boxes, that are then used by the control algorithm to perform the task at hand. This pipeline allows for easily developing control algorithms that leverage the power of deep learning for performing complex tasks. However, it is worth noting that such controllers are usually not optimal, since optimal control often requires anticipating scenarios that depend on higher level information that is not extracted from the data. For example, a drone covering an athletic event should slow down when following a runner that approaches the finish line, even if the runner accelerates towards reaching the end of the race. So, even though the bounding box of an athlete is usually enough for accurately following her/him through the race, it discards useful information that could be used to improve the control quality. These limitations can be addressed either by manually creating handcrafted features that can be used to exhaustively handle such scenarios, or by using deep reinforcement learning methods (DRL) that are capable of learning how to perform optimal control directly using the raw data and without requiring to design handcrafted intermediate features [11]. In this way, DRL allows for merging the information analysis and control processes into one unified model that can be trained in an end-to-end fashion towards achieving the desired task.

Indeed, DRL is capable of providing agents that operate directly on the raw input, e.g., pixels in the case of visual information, and learning how to perform various tasks, ranging from maneuvering vehicles [1], to using tools and cooperating to achieve their goals in complex environments [12]. However, DRL agents, in contrast with "traditional" DL models, cannot be trained using static datasets. Instead, they typically require using interactive simulation environments, through which the agents acquire experience and adapt their behavior in order to learn how to achieve their goals. That is, the agents interact with the environment, by performing various actions, and they collect a reward for each of them. Then, they are optimized, according to the collected experience, in order to maximize the expected reward. The quality of the learned agent critically depends on the quality of the simulation environment used for training. This is even more important in robotics-related applications, in which the agents are then deployed in the real world. Therefore, usually either realistic simulations environments are employed, with the hope that the distribution shift induced by transferring the agent from the simulation to the real world will not greatly impact its performance, or *sim2real* learning methods are used to smooth the transition to the real world [13].

In this work, we study the problem of controlling a drone that carries a camera in order to perform frontal view shooting of human subjects. This problem has been tackled with various approaches in the past, ranging from using face detectors and PID controllers [14], to developing DRL approaches for end-to-end control [15, 16], due to its importance for various emerging applications, such as active perception, human-robot interaction, aerial cinematography, etc. It is worth noting that applying DRL for this task is not straightforward, due to the lack of appropriate simulation environments. Existing approaches usually just employ static datasets that have been extended to support DRL, usually offering a very limited variety of view poses (since they are based on a limited collection of static images) [15], raising significant concerns on the behavior of DRL agents on less constrained environments.

The main contribution of this paper is the development of a realistic simulation environment, by employing a state-of-the-art graphics engine, that will allow for training DRL agents under more challenging open-world scenarios. A snapshot of the developed simulation environment is shown in Fig. 1. Note that in this study we employ a significantly more complex and challenging environment compared to previous approaches, e.g., [15], in order to examine the behavior of DRL methods in a open-world setting, allowing to better understand how DRL methods behave and bringing us one step close to deploying such approaches on real applications. It is worth noting that, to the best of our knowledge, this is the first study in which an open-world simulator is employed for training DRL agents for performing frontal view shooting without using a static posed dataset, demonstrating that DRL can be successfully used, even under this challenging setting. The developed DRL agent is trained using an established value-based DRL method and employs a specially designed reward shaping approach that was critical for successfully training the developed model. Note that reward shaping is often employed in various challenging RL problems, especially when they involve delayed and sparse rewards, to increase the performance of the agents [17]. An open-source implementation of the developed simulator is provided at `https://github.com/opendr-eu` to further accelerate research on developing DRL approaches.

The rest of the paper is structured as follows. First, the developed simulation environment, along with the proposed control agent and reward shaping approach are described in Section 2. Then, the experimental evaluation is provided in Section 3. Finally, conclusions are drawn in Section 4.

2. PROPOSED METHOD

In this Section we describe the developed simulation environment and the capabilities of the employed agent, e.g., available actions, ways of observing and interacting with the environment, etc. Then, we provide a concise description of the employed DRL method, as well as of the proposed reward shaping approach used for training the agent.

2.1. Simulation Environment

The simulation environment was developed using the Unreal Engine 4. The environment represents a city that consists of four blocks, as already shown in Fig. 1. Other than the four buildings, bus stops, trees and other smaller props are also included. Furthermore, fourteen distinct 3D models of people were employed and inserted at various spots of the city. The selected models correspond to people of various ethnic groups, with a variety of clothes and features, allowing to examine the behavior of DRL methods under these more challenging conditions. Ten human 3D models were used for training, while the rest of them were used for evaluating the performance of the agent on previously unseen subjects, as shown in Fig. 2.

A drone, equipped with an RGB camera, is also included in the simulation. The purpose of the developed agent is to appropriately control the drone in order to acquire frontal shots of the human models. Note that the drone can perform actions that only affect the position and orientation of the drone, while the orientation of the camera remains fixed, i.e., the agent does not control the gimbal on which the camera is mounted. Therefore, in order to acquire the desired shots, the agent must learn how to appropriately control the drone. The developed environment supports 9 different discrete actions:

1) *Forwards*: Move the drone towards the drone's look direction with speed $0.3m/s$ for $0.3s$,

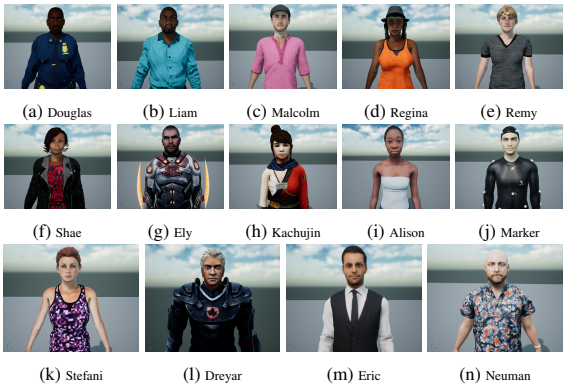


Fig. 2: Human models used in the developed simulation environment. The first ten (a-j) were used for the training process, while the rest of them (k-n) were used for evaluating the performance of the model. Note the large variations in style, clothing, and features exhibited by the employed human models.

2) *Backwards*: Move the drone in the opposite direction than the drone's look direction with speed $0.3m/s$ for $0.3s$,

3) *Left*: Move the drone to the left with respect to the drone's look direction with speed $0.3m/s$ for $0.3s$,

4) *Right*: Move the drone to the right with respect to the drone's look direction with speed $0.3m/s$ for $0.3s$,

5) *Up*: Move the drone up with speed $0.3m/s$ for $0.3s$,

6) *Down*: Move the drone down with speed $0.15m/s$ for $0.3s$,

7) *Rotate left*: Rotate the drone to the left with angular velocity $10°/s$ for $0.3s$,

8) *Rotate right*: Rotate the drone to the right with angular velocity $10°/s$ for $0.3s$,

9) *Stay*: Do not perform any action.

The above actions were implemented using the API provided by the AirSim plugin for Unreal Engine [18]. Also, note that the simulation was running 50 times faster than real time, so each second was equivalent to approximately 10 environment steps. Each episode lasts for 20 seconds, so about 200 steps are performed in each episode.

For every episode, one of the human models is selected at random, which will be the target human for the current episode. The selected human model is rotated to a random orientation so that the background appearing in the shot is different in each episode. The drone is placed in a random position in front of the human model's face, pointing at it. Some indicative examples of initial positions are shown in Fig. 3. Note again the wide variety of different backgrounds, compared to the uniform background observed by agents used

Fig. 3: Observations provided to the agent when an episodes begins. A 200×200 RGB frame is captured through the camera mounted on the drone at each simulation step.

Table 1: Neural network architecture used by the DRL agent

Layer	Output Shape
Input Layer	$200 \times 200 \times 3$
Convolutional Layer (8×8, stride 4)	$49 \times 49 \times 32$
Convolutional Layer (4×4, stride 2)	$24 \times 24 \times 64$
Convolutional Layer (3×3, stride 1)	$22 \times 22 \times 64$
Fully Connected Layer	512
Fully Connected Layer	9

in previous works (as shown in Fig. 1). The episode ends when either the human's face goes out of the frame, or the drone moves away from the target, or the drone collides with another object or finally when the 20 second time period is depleted from the start of the episode. The RL agent interacts with the developed environment and observes its state through the camera mounted on the drone, which provides one 200×200 RGB frame to the agent at each step. The agent is then trained to learn how to appropriately control the drone directly from this raw pixel input.

2.2. Deep Reinforcement Learning Agent

At each time step the agent observes a tensor $\mathbf{x} \in \mathbb{R}^{200\times200\times3}$ and decides which is the most appropriate action that should be selected in order to maximize the reward obtained from the environment. A value-based DRL approach is employed in this work: a deep learning model $f_{\mathbf{W}}(\mathbf{x}) \in \mathbb{R}^9$ is used to estimate the Q-value for each available action at each time step [19], where the notation $\mathbf{W}$ is used to denote the trainable parameters of the model. For any given observation, the agent selects the most profitable action by choosing the action with the largest Q-value. A fast and lightweight network architecture, that can also run on embedded processing platforms that drones typically use [20], was employed for this task. The architecture of the used network is shown in Table 1. The ReLU activation function [21] is used for all the convolutional and dense layers (except from the final one, which does not use any activation function and predicts the Q-values for the 9 possible actions). We also followed the dueling approach, proposed in [22], when designing the DRL agent.

A significant limitation of Q-learning is the instability of the learning process when non-linear functions, such as neural networks, are used to approximate the Q-values. To this end, in this work we used prioritized experience replay [23],

to increase the stability of the training process and reduce the correlation between the data used for training the model. The size of the experience replay pool is set to $N_{replay} = 500$ and batches of $N_{batch} = 32$ samples are drawn before each gradient descent update. Furthermore, we use a separate target network for generating the Q-values during the training to avoid feedback loops that can lead to instabilities. The target network is updated every $N_{target} = 500$ steps.

Finally, we used the Huber loss function for training the DRL model to regress the Q-values:

$$\mathcal{L}(\delta) = \begin{cases} \frac{1}{2}\delta^2, & \text{when } \delta < \delta_{thres} \\ |\delta| - \frac{1}{2}\delta_{thres}, & \text{otherwise} \end{cases}, \quad (1)$$

where δ is the error between the target Q-value and the current output of the network. The parameter δ_{thres} for the Huber loss was set to 1. For updating the weights of the network, the Adam optimizer [24] with learning rate $\lambda = 0.00005$ was used. Also, the discount factor γ was set to 0.95. Finally, to explore the solution space a linear exploration policy was used. Exploration starts with an initial rate of $\epsilon_{init} = 1$ and linearly decreases it to $\epsilon_{target} = 0.1$ during the first $N_{explore} = 1,900,000$ training steps. After the initial $N_{explore}$ steps the exploration rate stays constant to $\epsilon_{target} = 0.1$. The agent was trained for $2,000,000$ steps.

2.3. Reward Shaping

Defining a meaningful reward function is critical for training RL agents. The agent is trained to achieve the desired distance with respect to the selected human subject, as well as to acquire a frontal shot. Therefore, the optimization objective should involve both the distance to the target, as well as the angular error with respect to the human subject. The distance error is defined as:

$$d = \sqrt{(fp_x - cp_x)^2 + (fp_y - cp_y)^2 + (fp_z - cp_z)^2}, \quad (2)$$

where (fp_x, fp_y, fp_z) denotes the optimal position in which the drone should be in order to take a frontal close-up shot and (cp_x, cp_y, cp_z) is the current camera position. The angular error is similarly defined as:

$$\theta = \arccos\left(\frac{\mathbf{td}^T\mathbf{ld}}{||\mathbf{td}||_2||\mathbf{ld}||_2}\right), \quad (3)$$

where $\mathbf{td} = (td_x, td_y, td_z)$ is the person's face direction with respect to the drone's camera and $\mathbf{ld} = (ld_x, ld_y, ld_z)$ is the current look direction of the drone's camera. Therefore, the agents should be optimized with the respect to the combined distance and angular error.

However, simultaneously optimizing these two objectives when training an agent that directly operates on the raw camera input can be especially difficult [15]. To this end, we define an appropriate reward shaping approach that a) further rewards the agent as it gets closer to the target (by providing an additional r_p reward), b) encourages the agent to select the stay action (by further providing an additional $r_s > r_p$ reward), c) punishes the drone (by providing a negative reward equal to r_n) when collides with the another object or losses the human subject, d) disentangles the angle from the actual returned reward, as far as the agent stays within certain limits. Therefore, the reward function used in this paper is defined as:

$$r = \begin{cases} (1-d) + r_s + r_p, & \text{when } d < d_1 \text{ and } \theta < \theta_1 \text{ and the agent selects "Stay" (action 8)} \\ (1-d) + r_p, & \text{when } d < d_1 \text{ and } \theta < \theta_1 \\ -r_n, & \text{when } d > d_2 \text{ or } \theta > \theta_2 \text{ or the drone collides with another object} \\ 1-d, & \text{otherwise} \end{cases} \quad (4)$$

where for all the conducted experiments we set: $r_s = 90$, $r_p = 10$ and $r_n = 100$. The error bounds for the distance and angle can be set according to the requirements of each application. In this work, we set $d_1 = 0.1, \theta_1 = 0.1, d_2 = 1.13$ and $\theta_2 = 0.4$. Note that providing the additional r_s reward when the stay action is selected, allows not only to reach the target position and orientation, but also to train more robust agents that stay in place (select the "stay" action) when the desired shot is achieved and avoid erratic movements. In Section 3, we experimentally verified that this approach indeed leads to stabler agents that avoid performing back and forth movements just to slightly reduce the position error.

3. EXPERIMENTAL EVALUATION

The evaluation results are reported in Table 2. Two different evaluation setups were used: a) "train", where the ten human models that were used during the training process were used for the evaluation, and b) "test", where four different human models were used. For evaluating the method we ran 500 random episodes, where the agent was allowed to perform actions in a 20 second window. The proposed approach (denoted by the acronym "DRL" in Table 2) is also compared to a baseline agent that only selects the stay action. This allows to evaluate the improvements acquired by using the proposed agent. The total reward acquired through each episode is reported, as well the mean distance error (in meters). The proposed agent is indeed able to accurately control the agent to acquire a frontal shot, both for the train evaluation (the error is reduced from 0.60 to 0.15), as well as for the test evaluation (the error is reduced from 0.61 to 0.16). Note that the agent was capable of successfully generalizing the learned policy to subjects not seen during the training, as highlighted by its performance on the test evaluation.

Furthermore, to qualitative demonstrate the ability of the trained agent to control the trajectory of a drone in order to acquire a frontal shot, while avoiding erratic movements, we

Table 2: Drone control evaluation for frontal close-up shooting

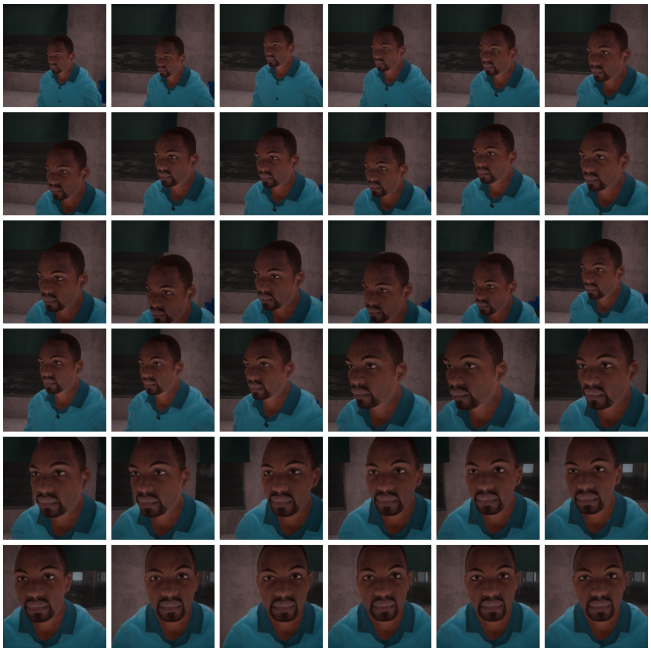
Method	Eval. Type	Mean Reward	Mean Error
Baseline	Train	140	0.604
DRL	Train	10,500	0.148
Baseline	Test	137	0.605
DRL	Test	6,250	0.164

provide an example of a control sequence in Fig. 4a. It is worth noting that after acquiring a satisfactory frontal shot, the agent selects the "stay" action, in order to acquire the reward associated with this action (r_s), demonstrating the effectiveness of the employed reward shaping approach. Also, in Fig. 4b, we also provide another control trajectory from a different viewing angle, showing both the human subject, as well as the drone. Again, similar conclusions can be drawn: the drone is appropriately controlled to lock its position in front of the human subject.

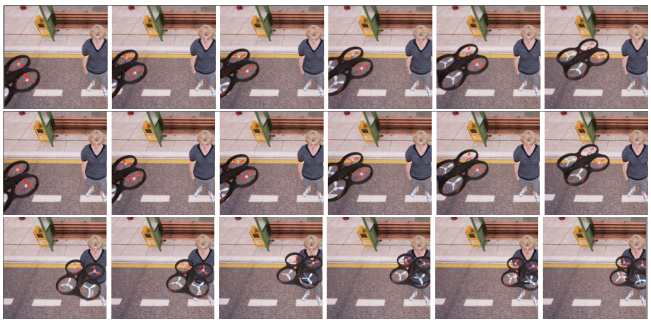
4. CONCLUSIONS

In this paper we presented a realistic simulation environment that can be used for developing deep reinforcement learning methods for various drone-related control tasks. The developed simulation environment goes beyond the static environments used by many existing methods, that largely employed datasets with little to no background variations. This open-world environment, that does not constraint the actions that can be performed, was used to develop an agent capable of controlling a drone, in order to perform frontal view shooting. The effectiveness of the developed approach was demonstrated using both subjects that were seen by the agent during the training, as well novel subjects, which were not seen during the training. The developed environment provides a solid step towards developing effective DRL methods for various robotics-related control tasks, allowing for easily implementing and experimenting with novel DRL approaches. To this end, we also provide an open-source implementation of the developing simulation environment, as well as of the DRL agent, allowing other researchers easily use and extent the methods presented in this paper.

There are several interesting future research directions. First, the proposed method can be used and evaluated in even more challenging settings, in which the human subjects interact with each other and potentially use other objects of the environment, e.g., cars. It is worth noting that this is a scenario that can be easily supported by the developed simulation environment. Furthermore, more advanced network architectures, such as recurrent neural networks, can be employed to ensure that the agents will capture the necessary temporal information needed for anticipating actions from each subject. Finally, the proposed method can be extended to control multiple axes, e.g., both the movement of a drone and the gimbal mounted on it, providing more flexible agents that can achieve shots of higher quality.



(a) View from the drone's camera



(b) View from a fixed point above the subject (the red rotors represent the front end of the drone)

Fig. 4: Two example control sequence of the trained DRL agent from two different perspectives

Acknowledgments: This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871449 (OpenDR). This publication reflects the authors' views only. The European Commission is not responsible for any use that may be made of the information it contains.

5. REFERENCES

[1] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy

search," in *IEEE International Conference on Robotics and Automation*, 2016, pp. 528–535.

[2] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[3] Mohammed Al-Qizwini, Iman Barjasteh, Hothaifa Al-Qassab, and Hayder Radha, "Deep learning algorithm for autonomous driving using googlenet," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2017, pp. 89–96.

[4] Nikolai Smolyanskiy, Alexey Kamenev, Jeffrey Smith, and Stan Birchfield, "Toward low-flying autonomous mav trail navigation using deep neural networks for environmental awareness," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017, pp. 4241–4247.

[5] A Claesson, D Fredman, L Svensson, M Ringh, J Hollenberg, P Nordberg, M Rosenqvist, T Djarv, S Österberg, J Lennartsson, et al., "Unmanned aerial vehicles (drones) in out-of-hospital-cardiac-arrest," *Scandinavian journal of trauma, resuscitation and emergency medicine*, vol. 24, no. 1, pp. 124, 2016.

[6] Tobias Nägeli, Lukas Meier, Alexander Domahidi, Javier Alonso-Mora, and Otmar Hilliges, "Real-time planning for automated multi-view drone cinematography," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, pp. 132, 2017.

[7] Nikolaos Passalis, Anastasios Tefas, and Ioannis Pitas, "Efficient camera control using 2d visual information for unmanned aerial vehicle-based cinematography," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2018, pp. 1–5.

[8] Karl Johan Åström, Tore Hägglund, and Karl J Astrom, *Advanced PID control*, vol. 461, The Instrumentation, Systems, and Automation Society Research Triangle, 2006.

[9] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[10] Naiyan Wang and Dit-Yan Yeung, "Learning a deep compact image representation for visual tracking," in *Proceedings of the Advances in Neural Information Processing Systems*, 2013, pp. 809–817.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529, 2015.

[12] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch, "Emergent tool use from multi-agent autocurricula," *arXiv preprint arXiv:1909.07528*, 2019.

[13] Fereshteh Sadeghi, Alexander Toshev, Eric Jang, and Sergey Levine, "Sim2real viewpoint invariant visual servoing by recurrent control," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4691–4699.

[14] Rita Cunha, Miguel Malaca, Vasco Sampaio, Bruno Guerreiro, Paraskevi Nousi, Ioannis Mademlis, Anastasios Tefas, and Ioannis Pitas, "Gimbal control for vision-based target tracking," in *Proceedings of the European Conference on Signal Processing*, 2019.

[15] Nikolaos Passalis and Anastasios Tefas, "Deep reinforcement learning for controlling frontal person close-up shooting," *Neurocomputing*, vol. 335, pp. 37–47, 2019.

[16] Nikolaos Passalis and Anastasios Tefas, "Continuous drone control using deep reinforcement learning for frontal view person shooting," *Neural Computing and Applications*, pp. 1–12, 2019.

[17] Andrew Y Ng, Daishi Harada, and Stuart Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the International Conference on Machine Learning*, 1999, vol. 99, pp. 278–287.

[18] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017.

[19] Hado Van Hasselt, Arthur Guez, and David Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[20] Nathan Otterness, Ming Yang, Sarah Rust, Eunbyung Park, James H Anderson, F Donelson Smith, Alex Berg, and Shige Wang, "An evaluation of the nvidia tx1 for supporting real-time computer-vision workloads," in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*, 2017, pp. 353–364.

[21] Xavier Glorot, Antoine Bordes, and Yoshua Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[22] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[23] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[24] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.