Deep Reinforcement Learning for Frontal View Person Shooting using Drones

Nikolaos Passalis and Anastasios Tefas
Department of Informatics, Aristotle University of Thessaloniki
Thessaloniki 54124, Greece
Email: passalis@csd.auth.gr, tefas@aiia.csd.auth.gr

***Abstract*—Unmanned Aerial Vehicles (UAVs), also known as *drones*, are increasingly used for a wide variety of novel tasks, including drone-based cinematography. However, flying drones in such setting requires the coordination of several people, increasing the cost of using drones for aerial cinematography and limiting the shooting flexibility by putting a significant cognitive load on the director and drone/camera operators. To overcome some of these limitation, this paper proposes a deep reinforcement learning (RL) method for performing autonomous frontal view shooting. To this end, a realistic simulation environment is developed, which ensures that the learned agent can be directly deployed on a drone. Then, a deep RL algorithm, tailored to the needs of the specific application, is derived building upon the well known deep Q-learning approach. The effectiveness of the proposed technique is experimentally demonstrated using several quantitative and qualitative experiments.**

I. Introduction

Unmanned Aerial Vehicles (UAVs), also known as *drones*, are increasingly used for a wide variety of novel tasks, ranging from fire detection in forests [1], and wildlife protection [2], to promptly responding to medical emergencies [3]. Drones are also used for cinematography tasks due to their high versatility and ability to capture spectacular aerial shots [4]. However, flying a drone in such settings requires the coordination of several people. A pilot has to control each drone, while a camera operator has to control the shooting camera on each drone. At the same time, the director has to coordinate several pilots and camera operators, if multiple drones are used, and ensure the quality of the captured shots. This situation increases the cost of using drones for aerial cinematography and severely limits the shooting flexibility by putting a significant cognitive load on the director and drone/camera operators.

The aforementioned limitations led to the development of various techniques for assisting drone-based cinematography, ranging from techniques for automated planning for multi-view drone shooting [4], and crowd avoidance for complying with drone legislation [5], to autonomous drone control systems [6], and various techniques for steering drone video shooting [7], [8]. These techniques automate various parts of the shooting process, reducing the cost of aerial cinematography and the cognitive load of human operators. However, we are still far from developing fully autonomous drones that would be able to automatically shot professional-grade footage according to a predefined plan, as designed by the director, while detecting salient events for performing opportunist shooting. This paper focuses on automating the drone control process for performing autonomous frontal view shots. That is, we aim to appropriately control the height and the relative position of the drone with respect to the person of interest to acquire a clear frontal shot.

Several approaches can be used to achieve this goal. Perhaps the most widely used is to project a set of known 2D landmark points, e.g., nose, eyes, mouth, etc., into the 3D space and solve the corresponding Perspective-n-Point (PnP) problem to estimate the pose of the head of the person of interest [9], [10]. Then, a PID controller can be used to appropriately control the drone to acquire the desired frontal shot [11], [12]. However, this approach requires accurately detecting several 2D facial landmark points, which can be especially difficult if a low-resolution image input is used. Note that this is usually the case, since drones frequently use a low-resolution video feed for performing the on-board processing tasks, even when high resolution cameras are available, due to their limited processing power and memory. Apart from these, careful calibration of the system is required and it is non-trivial to extend this approach for estimating the pose of other objects, since the landmark points have to be appropriately redefined and a corresponding detector must be developed.

Some of the aforementioned drawbacks can be addressed by using deep pose estimation algorithms that are directly trained to estimate the pose of various objects using a training set of data that contains images along with their pose annotations [13], [14]. Even though such deep learning approaches have been shown to increase the pose estimation accuracy and lead to more robust pose estimators, there is no guarantee that they will be *optimal* for performing control tasks. Developing optimal control algorithm has a long history in various engineering fields [15], [16]. Quite recently, it was shown that combining the great learning capacity of deep learning models with reinforcement learning (RL) techniques can lead to the development of robust control policies, that can work under stochastic and noisy environments, producing spectacular results, that often outperform humans on sophisticated control tasks [17], [18], [19]. However, RL techniques require realistic simulations of the environment where the RL agent will operate, which may not be always available.

The main contribution of this paper is the proposal of a deep RL method for accurately controlling a drone for performing frontal view shooting. To this end, we use the Head

Pose Image Database [20], to develop a realistic simulation environment that ensures that the learned agent can be directly deployed on a drone. Furthermore, we develop a deep RL algorithm tailored to the needs of the specific application, building upon the well known deep Q-learning approach [17], and we experimentally demonstrate the effectiveness of the proposed technique. We compare the proposed method to a controller that directly uses the output of a deep model that performs pose estimation and we demonstrate the superior behavior of the proposed RL method. Note that there are several recent deep RL approaches for various robotics applications [21], [22], [23]. However, to the best of our knowledge, this is the first work where deep RL is used to perform accurate drone control for frontal view shooting using a dedicated simulation environment. To further boost RL research, that critically relies on the availability of simulation environments, we provide an open-source implementation of the developed simulator as well as of the developed techniques at https://github.com/passalis/drone_frontal_rl.

The rest of the paper is structured as follows. The proposed method is described in detail in Section II. Then, the experimental evaluation is provided in Section III. Finally, future work is discussed and conclusions are drawn in Section IV.

II. Proposed Method

First, we provide a brief introduction to RL and the used notation. Then, the developed simulation environment for performing drone control for frontal shooting is presented. Finally, the proposed Deep RL method is derived and discussed in detail.

A. Deep Reinforcement Learning

Given an environment for which the *Markov property* is satisfied, i.e., the future state depends only on the current state and the selected action (or equivalently, given the current state and action the next state is conditionally independent of the previous states and actions), RL can be modeled using Markov Decision Processes (MDPs). A MDP can be defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}(\cdot), \mathcal{R}(\cdot), \gamma)$, where:

1) $\mathcal{S}$ is the set of possible states for the environment,
2) $\mathcal{A} = \{a_1, a_2, ..., a_{N_a}\}$ is the set of possible actions, where N_a is the number of possible actions,
3) $\mathcal{P}(s_{t+1}|s_t, a_t)$ is the probability that the environment will transit from the state s_t to the state s_{t+1} given that the action a_t has been performed,
4) $\mathcal{R}(s_t, a_t, s_{t+1}) = r_t$ is the *immediate* reward that the agent receives when performing the action a_t and transitioning from state s_t to state s_{t+1},
5) γ is a discount factor that defines the importance of immediate rewards versus future rewards, where typically $0 \leq \gamma \leq 1$. For $\gamma = 0$ the agent is *short-sighted*, while higher values increase the importance of future rewards.

An agent starts at state s_0 and selects the next action according to a *policy* $\pi(s)$ that defines the action that the agent will perform. Note that the policy $\pi(\cdot)$ can be stochastic and it is usually described as a probability distribution over the available actions, i.e., $\pi : \mathcal{S} \rightarrow \mathcal{P}(A)$. The agent's behavior defines a sequence of state-action-rewards $s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, ..., s_T$ that describes the history of the agent for a given episode that ends after T steps. The discounted accumulated return for an episode consisting of T steps is defined as:

$$R = \sum_{t=1}^{T} \gamma^{t-1} r_t, \tag{1}$$

where each action is selected according to the policy $\pi(s)$. RL aims to learn the optimal policy π^* that provides the maximum expected return:

$$\pi^* = \arg\max_{\pi} \mathbb{E}[R|\pi]. \tag{2}$$

Even though several approaches have been proposed for tackling this problem [24], in this paper the Q-learning method is used. In Q-learning the optimal action-value function $Q^*(s, a)$ is used to express the expected reward of performing the action a from the state s, given that an optimal policy is then followed. More formally, the action-value function $Q^*(s, a)$ is defined as:

$$Q^*(s, a) = max_{\pi} \mathbb{E}[R_t|s_t = s, a_t = a, \pi],$$

where R_t is the future discounted reward at time t defined as:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}. \tag{3}$$

In other words, the action-value function $Q^*(s, a)$ measures the maximum reward that an agent can earn after performing the action a and then using an optimal policy for the next transitions.

Bellman equation can be used to derive a method for learning the optimal action-value function starting from an initial guess $Q_1(s, a)$ [24]. Then, the action-value function is learned by observing the behavior of the environment, as the agent selects various actions, as:

$$Q_{i+1}(s_t, a_t) = (1-\eta) Q_i(s_t, a_t) + \eta(r_t + \gamma max_a Q_i(s_{t+1}, a)), \tag{4}$$

where η is the used learning rate. This algorithm, that belong to the family of *value iteration algorithms* [24], converges to the optimal action-value function $Q^*(s, a)$ as $i \rightarrow \infty$. Even though simple look-up tables can be used to represent and store the optimal values (Q-values), this approach quickly becomes impractical as the size of the state set $\mathcal{S}$ increases. To overcome this limitation, a deep neural network can be used to approximate the action-value function (deep Q-learning). The neural network is updated after each time-step using stochastic gradient descent [25], to minimize an appropriately defined loss function $\mathcal{L}(\cdot)$ that measures the error between the current estimation $Q(s, a, \mathbf{W}_i)$ and the target value provided by the Bellman equation, where $\mathbf{W}_i$ are the parameters of the deep neural network after i optimization iterations. The difference

(a) Face images at -60° tilt (pan varies from -90° to 90° in steps of 15°)

(b) Face images at -30° tilt (pan varies from -90° to 90° in steps of 15°)

(c) Face images at -15° tilt (pan varies from -90° to 90° in steps of 15°)

(d) Face images at 0° tilt (pan varies from -90° to 90° in steps of 15°)

(e) Face images at 15° tilt (pan varies from -90° to 90° in steps of 15°)

(f) Face images at 30° tilt (pan varies from -90° to 90° in steps of 15°)

(g) Face images at 60° tilt (pan varies from -90° to 90° in steps of 15°)

Fig. 1: Simulation environment: The drone moves in a sphere (approximately) centered at the face of the subject. The control commands (left/right/up/down) are simulated using the appropriate images from HPID.

between the current estimation and the value provided by the Bellman equation is defined as:

$$\delta = Q(s_t, a_t, \mathbf{W}_i) - (r_t + \gamma max_a Q(s_{t+1}, a, \mathbf{W}_{i-1}))), \quad (5)$$

and can be minimized using any appropriate loss functions, e.g., the squared loss $\mathcal{L}(\delta) = \frac{1}{2}\delta^2$. The proposed deep RL algorithm, that builds upon deep Q-learning [17], is analytically derived in Subsection II-C. The interested reader is also referred to [26], [27], [24], for a more in-depth review of RL.

B. Proposed Drone Control Environment

The Head Pose Image Database [20], abbreviated as "HPID" thereof, was used to develop the simulation environment. HPID contains 2,790 face images of 15 subjects in various poses taken in a constrained environment. More specifically, for each person face images with various head tilts (vertical angle) and pans (horizontal angle) were taken. For the tilt, head photos at $-90^\circ, -60^\circ, -30^\circ, -15^\circ, 0^\circ, 15^\circ, 30^\circ, 60^\circ$, and 90° were taken, while for the pan images that depict the head at $-90^\circ, -75^\circ, -60^\circ, -45^\circ, -30^\circ, -15^\circ, 0^\circ, 15^\circ, 30^\circ, 45^\circ, 60^\circ$, 75° and 90° were used. The developed simulator uses these images to simulate the movement of a drone in a part of a sphere defined by the center of the head of the subject. That allows for simulating the movement of a drone in 15° steps for the pan and in $15^\circ/30^\circ$ steps for the tilt. The various poses, obtained from different shooting angles for a person of the HPID, are shown in Fig. 1. The full pan range exists only for the images with tilt angles between -60° and 60°. Therefore, we restrict the available tilt angles used in the simulator to the aforementioned range.

The developed environment supports 5 different actions (assuming that the drone moves on the sphere defined previously):

1) *stay*: do not perform any action (to be used when a clear frontal view has been obtained),
2) *left*: move the drone left by $15^\circ \rightarrow$ pan decreases by 15°,
3) *right*: move the drone right by $15^\circ \rightarrow$ pan increases by 15°,
4) *up*: move the drone upwards by $15^\circ/30^\circ$ (depending on the available annotations) $\rightarrow$ tilt decreases by $15^\circ/30^\circ$, and
5) *down*: move the drone downwards by $15^\circ/30^\circ$ (depending on the available annotations) $\rightarrow$ tilt increases by $15^\circ/30^\circ$.

During these movements, we assume that the camera is appropriately controlled to keep the face centered, e.g., using a PID controller [12]. If an agent requests a control command that exceeds the limits of the simulator, e.g., an angle larger than 90°, then the simulator remains at its last state. An *OpenAI Gym*-compatible [28], environment was developed, and an open-source implementation is provided at

https://github.com/passalis/drone_frontal_rl.

C. Proposed Deep Reinforcement Learning Technique for Drone Control

In this Section the complete pipeline of the proposed deep RL technique for drone control is described. The RL agent interacts with the developed environment and *observes* its state, i.e., the acquired shot as shown in Fig. 1. To simplify the learning process, we use a face detector to detect and crop the face image [29]. For the conducted experiments, we directly used the head annotations supplied by HPID. After appropriately cropping the image, it is resized to 64×64 pixels. Therefore, the RL agent at the t-th time step observes a tensor $\mathbf{x}_t \in \mathbb{R}^{64 \times 64 \times 3}$ that corresponds to the cropped face image.

Defining a meaningful reward function is critical for the fast and stable convergence of RL algorithms. Even though RL can deal, to some extent, with sparse and time-delayed rewards, we experimentally found out that rewarding (or punishing) the agent after each action can significantly speed up the learning process. To define the reward function we first have to define the control error at the t-th step:

$$e_t = \frac{1}{2}((x_t/60)^2 + (x_p/90)^2), \tag{6}$$

where x_t is the current tilt (in degrees) and x_p is the current pan (in degrees). To acquire non-frontal shots, this error must be appropriately modified to express the error around the desired tilt and pan angle. Then, the reward function is defined as:

$$r_t^{(raw)} = \begin{cases} 0, & \text{if } e_t > e_{thres} \\ 1 - e_t/e_{thres}, & \text{otherwise} \end{cases}, \tag{7}$$

where e_{thres} is the threshold for rewarding the agent. If e_{thres} is set to 1, then the agent is rewarded at every time-step. Even though the reward is proportional to the control error e_t, this can slow down the learning process. In the conducted experiments we only reward the agent when it acquires the correct frontal shot, i.e., e_{thres} was set to 0.05. To further boost the learning process, we provide an extra small reward/punishment whenever the agent makes a correct/wrong movement:

$$r_t^{(bonus)} = \begin{cases} -0.15, & \text{if } e_t > e_{t-1} \\ 0.1, & \text{if } e_t < e_{t-1} \\ 0, & \text{otherwise} \end{cases}. \tag{8}$$

Using a slightly higher penalty for wrong actions ensures the stability of the control process and discourages control oscillations. Therefore, the final reward function is defined as the sum between the raw reward $r_t^{(raw)}$, which depends on the control error e_t, and the "bonus" reward $r_t^{(bonus)}$, that encourages correct and stable control actions:

$$r_t = r_t^{(raw)} + r_t^{(bonus)}. \tag{9}$$

To approximate the action-value function we used a deep convolutional network. A fast and lightweight network architecture, that can run on-drone and it is composed of less than 70,000 parameters, was used for this task. The architecture of the proposed network is shown in Table I. Batch normalization is used after each convolutional layer [30], while the *relu* activation function 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 5 possible actions) [31].

TABLE I: Neural network architecture

Layer Type	Output Shape
Input	$64 \times 64 \times 3$
Convolutional (5×5, stride 2)	$30 \times 30 \times 16$
Batch Normalization	$30 \times 28 \times 30$
Max Pooling (2×2)	$15 \times 15 \times 16$
Convolutional (3×3)	$13 \times 13 \times 32$
Batch Normalization	$13 \times 13 \times 32$
Max Pooling (2×2)	$6 \times 6 \times 32$
Convolutional (3×3)	$4 \times 4 \times 64$
Batch Normalization	$4 \times 4 \times 64$
Max Pooling (2×2)	$2 \times 2 \times 64$
Dense	128
Dense	64
Dense	5

A significant problem in Q-learning is the instability of the learning process when non-linear functions, such as neural networks, are used to approximate the Q-values. Several techniques have been proposed to overcome this issue [26], [27]. In this work we use experience replay [32], that allows for reducing the correlation between the training data and improves the learning stability by using training instances from various timesteps and episodes, to address this issue. 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. This technique is known as "Double Q-learning" [33]. The target network is updated every $N_{target} = 500$ steps.

For the optimization procedure the Huber loss function is used:

$$\mathcal{L}(\delta) = \begin{cases} \frac{1}{2}\delta^2, & \text{if } \delta < \delta_{thres} \\ |\delta| - \frac{1}{2}\delta_{thres}, & \text{otherwise} \end{cases}, \tag{10}$$

where δ is the difference error defined in Eq. (5). Huber loss is used instead of squared loss, since it is more robust to outliers, providing smoother gradients and stabilizing the learning process (δ_{thres} was set to 1). This is especially important when the reward can accumulate to relatively large values, e.g., 50, as in the used environment. For updating the weights of the network, the RMSProp optimizer with learning rate $\eta = 0.00005$ was used [34]. 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 $N_{explore} = 900,000$ training steps. After the initial $N_{explore}$ steps the exploration rate stays constant to $\epsilon_{target} = 0.1$. During the testing/evaluation a small exploration

rate is also used, i.e., $\epsilon_{test} = 0.05$. The agent was trained for 1,000,000 steps that correspond to 20,000 control episodes. For each episode a random initial position was used for the drone. For training, head images from 10 persons of the HPID were used, while for evaluating the proposed method images from the rest 5 persons were used. The keras-rl library was used to implement the proposed method [35].

III. Experimental Evaluation

The evaluation results are reported in Table II. Two different evaluation setups were used: a) "train", where the 10 persons that were used during the training process were used for the evaluation, and b) "test", where 5 different persons were used. For evaluating the method we ran 500 random episodes, where for each episode the agent was allowed to perform 20 control actions. The proposed method is abbreviated as "D-RL" in Table II. The learned agent was also compared to two other strategies: a) using a dummy agent that does not perform any control action (abbreviated as "Stay") and b) using a deep CNN to perform pose regression and then appropriately control the drone (abbreviated as "Pose Regressor"). The Pose Regressor network use the same architecture as the CNN used for estimating the Q-values (Table I) and was trained to directly regress the tilt and pan of a face image using the same train/test setup, leading to a mean tilt error of 16.67° and a mean pan error of 13.71° (evaluation on the test set). The proposed RL method outperforms the Pose Regression technique for both the train and test setups, demonstrating the importance of learning optimal controllers for the task at hand instead of relying on hand-crafted methods to perform control.

The reward and the mean Q-value during the training process are shown in Fig. 2. During the exploration phase (episodes 0-18,000) the mean reward increases and stabilizes during the last 2,000 episodes, where the exploration rate is fixed to 0.1. However, note there are a few episodes that were not solved correctly during the last 50,000 training steps. This issue can be probably solved by performing more optimization iterations and/or using larger exploration during the final steps to ensure that the agent will learn a more effective control policy. The mean Q-value increases during the training, as expected, and quite smoothly converges without exploding. Furthermore, the performance of the agent during various training checkpoints (every 100,000 training steps) are shown in Table III. The best training performance is obtained during the 800,000-th training step and this model was chosen to perform the evaluations. The slight decrease in the performance of the agent after the 800,000-th training step can be possibly attributed to the decreased exploration rate.

Finally, several control sequences are shown in Fig. 3 (train sequences) and Fig. 4 (test sequences). Several conclusions can be drawn from these sequences. First, the proposed method is able to accurately control the simulated drone to acquire a frontal view shot both for the train and test evaluations. Even though in most cases the agent is capable of recognizing when the frontal shot has been obtained and emit the "stay" command, the relatively high reward, that is given when the absolutely correct pose has been achieved, can sometimes lead to oscillations around the frontal position, e.g., row 4 of Fig. 3. Such oscillations are even more frequent in the test evaluation. This problem can be addressed by using a different reward function that penalizes this kind of behavior.

IV. Conclusions

In this work we presented and evaluated a deep RL method for performing drone control to acquire high-quality frontal view person shots. The proposed method builds upon the well established deep Q-learning approach [17], and was experimentally demonstrated that it is capable of performing accurate control. Furthermore, the proposed technique was able to outperform a hand-crafted controller, that uses a dedicated deep face pose estimator to control the drone, highlighting the importance of learning optimal controllers instead of relying on hand-crafted control techniques.

Several interesting future research direction exist. First, the impressive performance of the proposed technique paves the way for several other deep RL-based control methods for autonomous drone-based cinematography. Furthermore, the proposed method can be applied to more dynamic environments, e.g., moving targets can be used, and more sophisticated deep architectures, e.g., advanced pooling layers [36], or recurrent networks [37], can be employed to more accurately model the target and control the camera. Furthermore, the use of end-to-end trainable systems, that simultaneously perform camera and drone control, can be developed and evaluated. Finally, the development of transfer learning techniques for RL, that can combine the use of real datasets and simulators, that use computer-generated graphics, will allow for training RL techniques under a wider range of scenarios, while ensuring that they can be directly deployed in real-world applications.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE). 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.

References

[1] W. Krüll, R. Tobera, I. Willms, H. Essen, and N. von Wahl, "Early forest fire detection and verification using optical smoke, gas and microwave sensors," *Procedia Engineering*, vol. 45, pp. 584–594, 2012.

[2] A. Gynnild, "The robot eye witness: Extending visual journalism through drone surveillance," *Digital journalism*, vol. 2, no. 3, pp. 334–343, 2014.

[3] 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, p. 124, 2016.

[4] T. Nägeli, L. Meier, A. Domahidi, J. Alonso-Mora, and O. Hilliges, "Real-time planning for automated multi-view drone cinematography," *ACM Transactions on Graphics*, vol. 36, no. 4, p. 132, 2017.

[5] M. Tzelepi and A. Tefas, "Human crowd detection for drone flight safety using convolutional neural networks," in *Proceedings of the European Signal Processing Conference*, 2017, pp. 743–747.

TABLE II: Drone control evaluation for frontal shooting

Method	Evaluation Type	Mean Reward	Mean Absolute Tilt Error (°)	Mean Absolute Pan Error (°)
Stay	Train	-	29.94	48.24
Pose Regressor	Train	-	6.87	7.44
D-RL	Train	15.63	**1.26**	**4.38**
Stay	Test	-	29.37	48.93
Pose Regressor	Test	-	14.07	12.18
D-RL	Test	7.40	**12.12**	**11.34**

(The absolute mean pan and tilt error (to obtain a perfectly frontal shot) are reported.)

(a) Reward during the training

(b) Reward during the last 1000 training episodes

(c) Mean Q-value during training

Fig. 2: D-RL: Learning curves

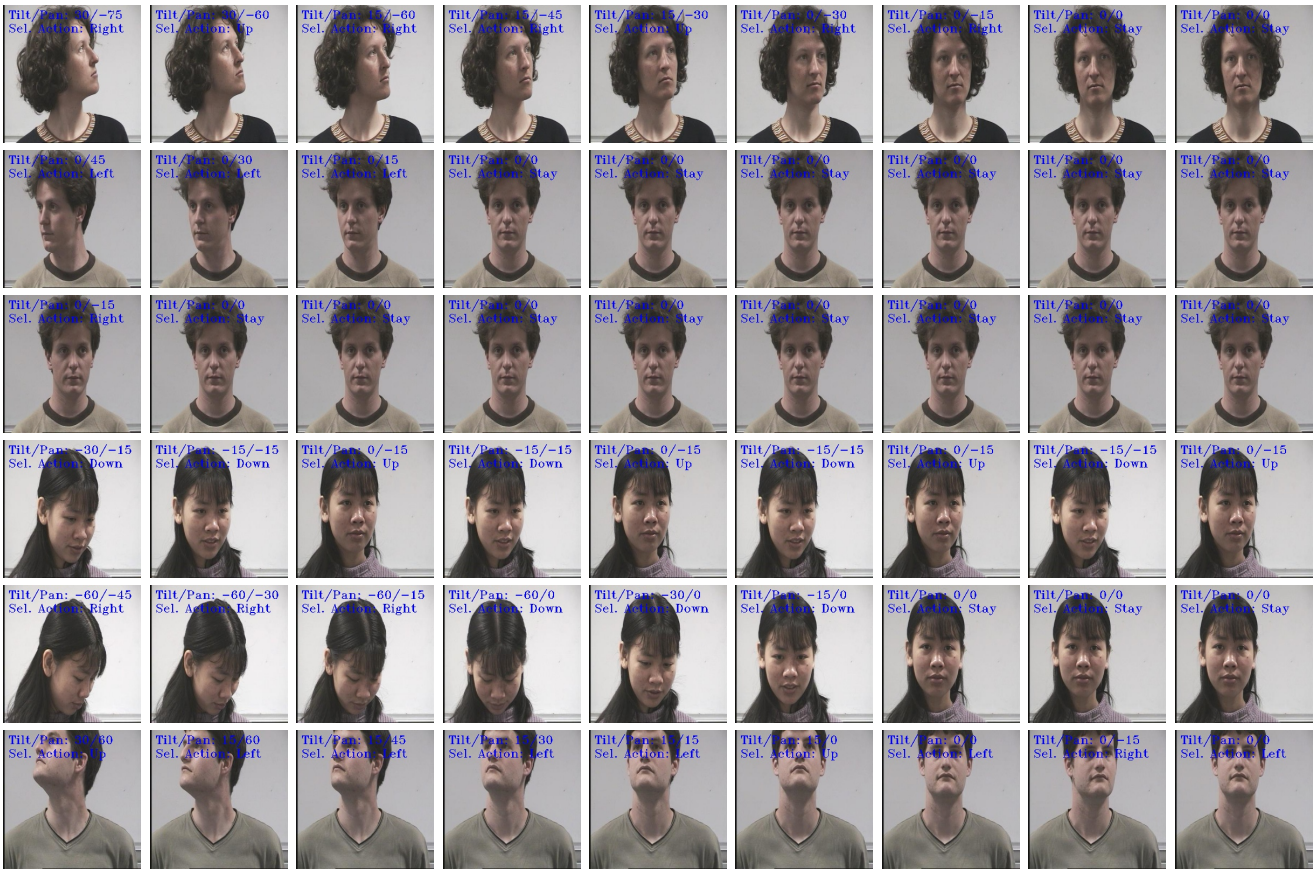

Fig. 3: Train control sequences

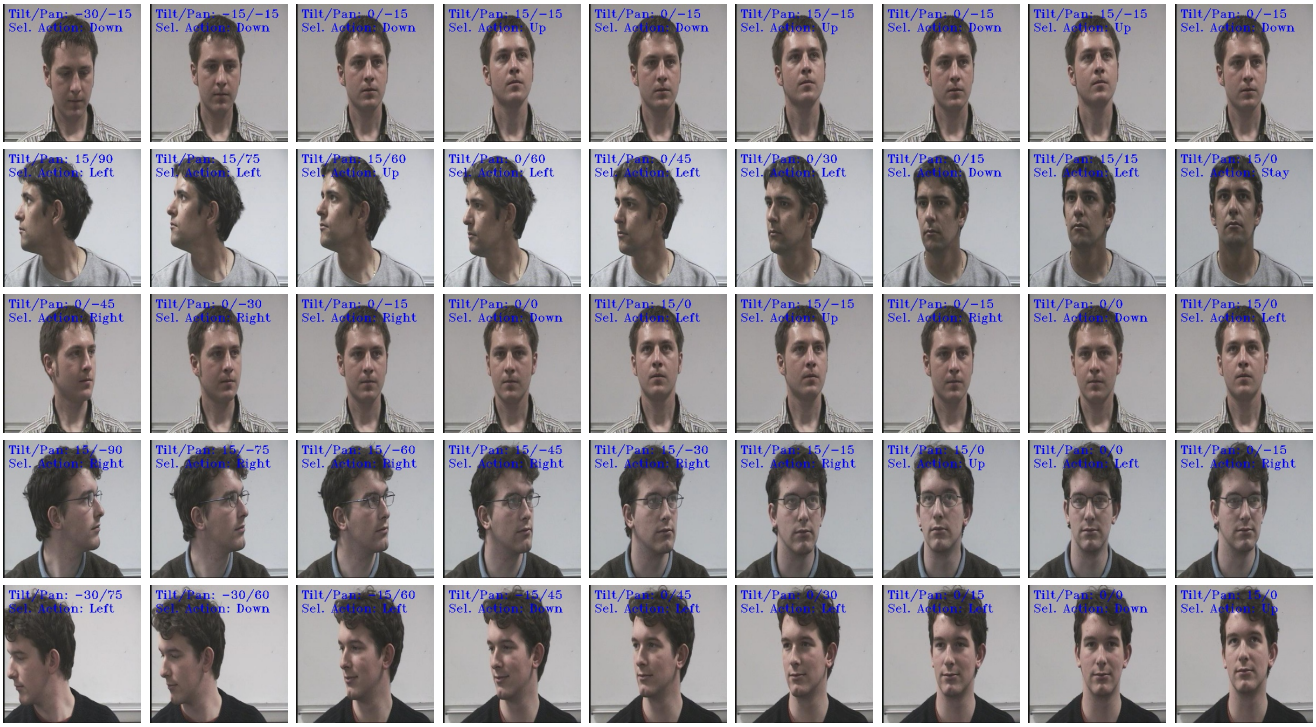

Fig. 4: Test control sequences

TABLE III: Performance of the RL agent during the training

Training Step	**Mean Abs Reward**	**Tilt Error** (°)	**Pan Error** (°)
Initial	0.46	42.63	47.85
100,000	8.74	13.83	17.58
200,000	9.07	7.77	21.45
300,000	12.34	6.09	14.13
400,000	12.32	8.94	10.92
500,000	13.73	5.25	10.62
600,000	13.03	4.77	12.57
700,000	13.71	5.73	7.83
800,000	**15.63**	**1.26**	**4.38**
900,000	14.67	3.54	6.67
1,000,000	14.21	5.70	8.67

(The absolute mean pan and tilt error (to obtain a perfectly frontal shot) are reported.)

[6] Q. Galvane, J. Fleureau, F.-L. Tariolle, and P. Guillotel, "Automated cinematography with unmanned aerial vehicles," *arXiv preprint arXiv:1712.04353*, 2017.

[7] N. Passalis and A. Tefas, "Concept detection and face pose estimation using lightweight convolutional neural networks for steering drone video shooting," in *Proceedings of the European Signal Processing Conference*, 2017, pp. 71–75.

[8] ——, "Improving face pose estimation using long-term temporal averaging for stochastic optimization," in *Proceedings of the International Conference on Engineering Applications of Neural Networks*, 2017, pp. 194–204.

[9] V. Kazemi and S. Josephine, "One millisecond face alignment with an ensemble of regression trees," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1867–1874.

[10] S. Ohayon and E. Rivlin, "Robust 3d head tracking using camera pose estimation," in *Proceedings of the International Conference on Pattern Recognition*, vol. 1, 2006, pp. 1063–1066.

[11] K. H. Ang, G. Chong, and Y. Li, "Pid control system analysis, design, and technology," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 4, pp. 559–576, 2005.

[12] N. Passalis, A. Tefas, and I. Pitas, "Efficient camera control using 2d visual information for unmanned aerial vehicle-based cinematography," in *Proceedings of the International Symposium on Circuits and Systems*, 2018.

[13] A. Toshev and C. Szegedy, "Deeppose: Human pose estimation via deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1653–1660.

[14] R. Ranjan, V. M. Patel, and R. Chellappa, "Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[15] A. E. Bryson, *Applied optimal control: optimization, estimation and control*. CRC Press, 1975.

[16] D. E. Kirk, *Optimal control theory: an introduction*. Courier Corporation, 2012.

[17] 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*, vol. 518, no. 7540, p. 529, 2015.

[18] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning." in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 16, 2016, pp. 2094–2100.

[19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[20] N. Gourier, D. Hall, and J. L. Crowley, "Estimating face orientation from robust detection of salient facial features," in *ICPR International Workshop on Visual Observation of Deictic Gestures*. Citeseer, 2004.

[21] C. Finn, T. Yu, J. Fu, P. Abbeel, and S. Levine, "Generalizing skills with semi-supervised reinforcement learning," *arXiv preprint arXiv:1612.00429*, 2016.

[22] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2017, pp. 2786–2793.

[23] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with large-scale data collection," in *Proceedings of the International Symposium on Experimental Robotics*, 2016, pp. 173–184.

[24] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[25] S. Haykin and N. Network, "A comprehensive foundation," *Neural networks*, vol. 2, no. 2004, p. 41, 2004.

[26] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[27] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[28] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[29] D. Triantafyllidou, P. Nousi, and A. Tefas, "Fast deep convolutional face detection in the wild exploiting hard sample mining," *Big Data Research*, 2017.

[30] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the International Conference on Machine Learning*, 2015, pp. 448–456.

[31] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[32] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[33] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, pp. 2094–2100. [Online]. Available: http://dl.acm.org/citation.cfm?id=3016100.3016191

[34] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[35] M. Plappert, "keras-rl," https://github.com/matthiasplappert/keras-rl, 2016.

[36] N. Passalis and A. Tefas, "Learning bag-of-features pooling for deep convolutional neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 5755–5763.

[37] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.