2022 International Conference on Unmanned Aircraft Systems (ICUAS)
Dubrovnik, Croatia. June 21-24, 2022

Fighter Pilot Behavior Cloning

Viktor Sandström, Linus Luotsinen, Daniel Oskarsson
Swedish Defence Research Agency (FOI)

***Abstract*—In this paper, a feed-forward neural network is trained on a small dataset of human fighter pilot data, recorded from maneuvering a fixed-wing fighter aircraft in a flight simulator. The goal is to model the pilot behavior, using a technique called behavior cloning. By carefully preprocessing the training data, it is shown that this simple and intuitive approach results in a model that can successfully fly the aircraft at high velocity on flight tracks that demand sharp turns, and even perform maneuvers not explicitly represented in the data. Furthermore, it is demonstrated that a pretrained neural network will adapt to a significant change in flight dynamics with less training, compared to a previously untrained model. This transfer learning scenario is important since fine-tuning pretrained models could simplify the development of a wide fleet of AI aircraft.**

***Index Terms*—Imitation Learning, Transfer Learning, Behavior Cloning, Deep Learning, Fixed-Wing Aircraft, Unmanned Aerial System.**

I. Introduction

Air force pilots must continuously practice to be ready for any possible situation or rapid deployment. Much of the pilots practice time is spent in advanced flight simulator facilities designed for human-vs-human and human-vs-computer air combat. However, the *computer generated forces* (CGFs) in these facilities should be human-like in their decision-making, a characteristic that can be difficult to achieve [1]. Techniques such as *reinforcement learning* have shown performance surpassing human fighter pilots in air combat [2], which is not always desirable in such practice sessions. A potential solution is to use *imitation learning* to model the complex human pilot behavior needed to achieve realistic opponents. Imitation learning is the study of methods for developing models that can imitate demonstrations performed by an *expert* in some environment. These demonstrations are generally a sequence of state-action pairs, i.e., each occurring situation is labeled with the action taken by the expert. Given a dataset of state-action pairs recorded from the expert, the basic approach is to train a model, often called a *learner policy*, to predict the correct action at each state encountered by the expert. This simple and intuitive approach is often called *behavior cloning*. However, behavior cloning is not guaranteed to perform well since states encountered by the learner policy are not always represented in the training data. A small action error can lead to an unseen state, which would cause the learner policy to make a new error and thus ending up in an even worse state, and so on. To ensure high performance, the training data must contain a diverse set of states and actions [3], [4].

In this paper, the learner policy controls the aircraft in the same way as a human pilot would, i.e., control signals are sent directly to the cockpit joystick before passing into the flight simulator. The simulator is treated as a black box from which it is only possible to extract the current state and input joystick values.

In addition, it is of interest to investigate to what extent trained policies can be reused in other flight simulators. Imitation learning techniques require a flight simulator adapted or designed for data collection, training and execution of AI-models. However, adapting a military flight simulator designed with human training in mind is time-consuming and costly. Instead, one could develop learner policies in separate, more easy-to-use, flight simulators and exploit *transfer learning* to integrate them in the desired simulator. It is assumed that the main dissimilarity between two simulators are the underlying *flight dynamics model* (FDM), i.e., the mathematics and numerical methods used for calculating the dynamics of the aircraft. To investigate this idea, experiments are conducted where a policy, capable of flying in one FDM, is fine-tuned on new expert data originating from a significantly different FDM. The contributions of this research are mainly:

1) It is shown that a good state vector and a well-tuned training setup can result in satisfactory performance using behavior cloning, without the need for complex algorithms such as SMile [5], DAgger [6] or MwDAgger [7].
2) It is shown that a fine-tuned model can adapt to a different FDM with less training compared to a previously untrained model. It is thus likely that transfer learning can be used to more efficiently develop good performing models.

The assumption is that guidance and navigation remains unchanged with a change in dynamics, and that the control must be retrained to compensate for the difference in dynamic response to input.

II. Related work

Intelligent CGFs for air-combat have been studied extensively. Traditionally, CGFs have been scripted, which is a time-consuming process requiring deep domain-specific knowledge. Other techniques such as dynamic scripting [8], [9], and reinforcement learning [10], [11] have shown promising results, but often using simple two-dimensional flight dynamics models. In recent work [2], a reinforcement learning model, based on the maximum-entropy reinforcement algorithm *soft actor critic* (SAC) [12], managed to win against a real fighter pilot in a 1-vs-1 dogfight scenario using

978-1-6654-0593-5/22/$31.00 ©2022 IEEE

a high-fidelity 6-DOF flight dynamics model - a state-of-the-art result, but not necessarily desirable for use in basic fighter pilot training sessions.

Imitation learning has been used in a wide range of applications such as autonomous driving [13]–[15], first-person shooter games [16], and autonomous flight [17]–[20], but less research has focused on air-combat related problems. Much of the work on autonomous flight use neural networks to model parts of the full control system. In the 2017 work [21], fourteen separate neural networks were trained to imitate expert demonstrations on tasks such as controlling the landing gear, the rudder and maintaining altitude to enable a fully functional intelligent autopilot system. A behavior tree-like structure determines when certain networks are deployed and the full model successfully completes a full simulated flight, including taxi, take-off and landing.

In a more recent work [22], using a control theory approach, three neural networks were trained to map a time-window of states to control input signals. Each neural network was trained on different data to control certain situations, and a switch mechanism determines how much of each network to use in each timestep, calculated as a weighted sum. This approach successfully imitates an autopilot model, implemented in the high-fidelity flight simulator X-Plane, where the task was to reach certain headings and altitudes.

In a recent work, one neural network was trained to fly a fixed-wing aircraft on a square flight track [7]. The authors trained the model to map inputs containing waypoints, velocities and rotation angles to control inputs on a Skyhunter aircraft simulated with a non-linear flight dynamics model with six degrees of freedom (6-DOF). A labeled dataset was recorded from an autopilot and the authors show that standard supervised learning is *not* sufficient for their setup. They instead develop a sophisticated algorithm called *"Moving-Window DAgger"* with successful results.

The work in this paper is related to the work by [7], but differs in the following ways:

1) The policy was trained on human pilot data and not data from an autopilot. In this case, the aircraft was flown by the authors (none of which are pilots). The focus was not to perform skilled pilot maneuvers, but rather to capture a human-like flight behavior from data.
2) A different approach was used regarding training and data preprocessing. A larger network architecture, different optimizer and data augmentation techniques were used.
3) The input state vector was engineered to be direction-invariant, which reduces the amount of data needed to get satisfactory performance.

III. Preliminaries

Let π^* denote a deterministic expert policy, and let $\pi_\theta \in \Pi$ denote a deterministic learner policy parameterized by θ, where Π is a policy class. The learner policy is a function that maps states $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^d$ to actions $\boldsymbol{u} \in \mathcal{U} \subseteq \mathbb{R}^m$. The goal of imitation learning is often formulated as finding the optimal policy that minimizes the expected immediate loss,

$$\hat{\pi}_\theta = \underset{\pi_\theta \in \Pi}{\arg\min}\, \mathbb{E}_{\boldsymbol{x} \sim \rho_{\pi_\theta}} \left[\mathcal{L}(\pi^*(\boldsymbol{x}), \pi_\theta(\boldsymbol{x})) \right], \tag{1}$$

where $\mathcal{L} : \mathcal{U} \times \mathcal{U} \to [0, \infty)$ is a loss function that quantifies the error in the action taken by the learner policy at state $\boldsymbol{x}$. Regularization is omitted for brevity. The distribution ρ_{π_θ} is dependent on the unknown and complex system dynamics as well as the policy itself. This dependency violates the independent and identically distributed (i.i.d) assumption that most statistical learning methods are based upon [23]. Solving this non-convex optimization problem is generally difficult and can be approximated by a reduction to supervised learning. The simplest of such approximations is behavior cloning.

A. Behavior Cloning

Behavior cloning is an approach that ignores the difficulty of handling ρ_{π_θ} and instead finds a policy that minimizes the expected immediate loss under the distribution of states induced by the expert, ρ_{π^*}. The new optimization problem can be formulated as

$$\hat{\pi}_\theta = \underset{\pi_\theta \in \Pi}{\arg\min}\, \mathbb{E}_{\boldsymbol{x} \sim \rho_{\pi^*}} \left[\mathcal{L}(\pi^*(\boldsymbol{x}), \pi_\theta(\boldsymbol{x})) \right], \tag{2}$$

which can be approximated by sampling expert demonstrations and applying a supervised learning algorithm. Note that in this setting, a low training, validation or test loss does not necessarily imply good policy performance, only that the policy can imitate expert actions well for the states close to those encountered in training. This is due to the mismatch in distribution when training and testing the model. Optimally, the model would be tested in the environment during training to validate the true performance.

The advantage of behavior cloning is that it is simple to implement and computationally cheap. Other more intricate methods, such as SMile [5] or DAgger [6], often involve a supervised learning step as a subroutine in a more complex algorithm. The disadvantage of behavior cloning is that the mismatch between distributions can have a negative effect on performance.

B. Transfer Learning

Transfer learning is a large and active research area focusing on methods for utilizing experience gained from a previously solved (source) problem, when considering a different related (target) problem. If the experience from the source is leveraged in a good manner, a better performing model can be obtained using less data and less training on the target problem [24]. Many transfer learning techniques exist, such as domain adaptation [25], domain randomization [26] and meta-learning [27], but the most basic, and widely used, technique is called fine-tuning [28]. Fine-tuning works by taking a subset $\tilde{\theta} \subseteq \theta$ of the trained parameters of the model and retraining them on the target problem. Selecting an optimal subset of parameters is not trivial, and the common approach is to either fine-tune the last few layers or the

full model. If there is some similarity between the source and target problem, it is possible achieve high performance more efficiently. This is standard practice in deep learning to achieve good performance with small datasets or short training sessions.

C. Aircraft Motion

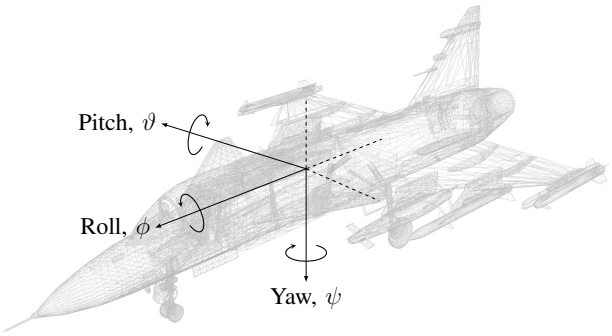


Fig. 1: Illustration showing Tait–Bryan angles on a Jas 39 Gripen. Right hand rule applies to all rotations.

When modelling flight dynamics, it is common to view the aircraft as a 6-DOF rigid body. This means that the aircraft can translate and rotate about three perpendicular directions. These rotations are said to occur around three axes called the *yaw-, pitch-* and *roll-axes*, with the origin placed at the aircraft's center of gravity [29], as depicted in Fig. 1. The roll axis is directed towards the nose and parallel to the line connecting the nose and the tail of the aircraft. Motion about this axis is called roll and is controlled primarily by the *ailerons*, a control surface on the wings. A positive roll lowers the right wing and forms the roll angle, $\phi \in [-\pi, \pi]$ between the roll axis and the surface of the earth. The pitch axis is parallel to the line drawn from wingtip to wingtip. Pitch is controlled by another control surface called the *elevator*. A positive pitch raises the nose of the aircraft and forms the pitch angle, $\vartheta \in [-\pi/2, \pi/2]$ with respect to the surface of the earth. Finally, the yaw axis points down perpendicular to the pitch and roll axes. Motion about this axis is called yaw and is controlled by the *rudder*. A positive yaw moves the nose of the aircraft to the right and the corresponding yaw angle is the angle between the global y-axis (north) and the projection of the local x-axis (forward) onto the surface of the earth. The yaw angle is denoted as $\psi \in [-\pi, \pi]$. These angles are commonly called the Tait-Bryan angles and are a special case of the well-known Euler angles. The motion of the aircraft is calculated using a set of parametric equations modelling lift, drag and thrust in each timestep. Such models can vary in significant ways depending on the underlying assumptions of aerodynamic forces, drag coefficient values at different velocities, et cetera.

D. Transformations

The Tait-Bryan angles can be used to transform any vector from a global (inertial) reference frame to a local reference frame [30]. Using the definitions of yaw, pitch and roll from above, the transformation $T : \mathbb{R}^3 \to \mathbb{R}^3$ can be represented by the orthonormal matrix

$$T(\phi, \vartheta, \psi) := T_\phi T_\vartheta T_\psi T_0, \tag{3}$$

where

$$T_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}, T_\vartheta = \begin{bmatrix} \cos\vartheta & 0 & -\sin\vartheta \\ 0 & 1 & 0 \\ \sin\vartheta & 0 & \cos\vartheta \end{bmatrix},$$

$$T_\psi = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{ and } T_0 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}.$$

The last matrix aligns the local reference frame to a north-east-down (NED) frame, such that the aircraft initially is flying north (aligning global y-direction with local x-direction)

IV. Learning to Fly

To test behavior cloning in the context of flying a fighter jet, the first objective is to develop a policy capable of flying a non-linear 6-DOF Jas 39 Gripen FDM to specified waypoints on a track (as specified in section IV-A). The simulator is Virtual BattleSpace 3 (VBS3), a military simulator software that includes a flight simulator module. A control loop was developed to enable flight data extraction in real-time and allow for policy control by injecting control signals into the simulator.

Fig. 2: Cockpit view from the VBS3 Jas 39 Gripen. The task is to navigate to the cross which marks the next waypoint. For the expert, this view is the state at time t. The policy receives a feature vector as explained below.

The expert receives a state input in the form of visual graphics, and steers the aircraft via a manual joystick which is mapped to a virtual joystick software. The cockpit view is depicted in Fig. 2. The policy receives a feature vector, ten times per second, which represents the current state. As a response to each state, a control signal is sent to the same

virtual joystick software. See Fig. 3 for a schematic view of the setup. It is important to highlight that, even though the policy is deterministic, the sampling of flight data from the simulator is not. This means that two identical starting positions may result in two different trajectories.

The second objective is to swap the FDM and use fine-tuning to investigate if a pretrained policy learns the new setting with less data compared to a randomly instantiated policy on the same task.

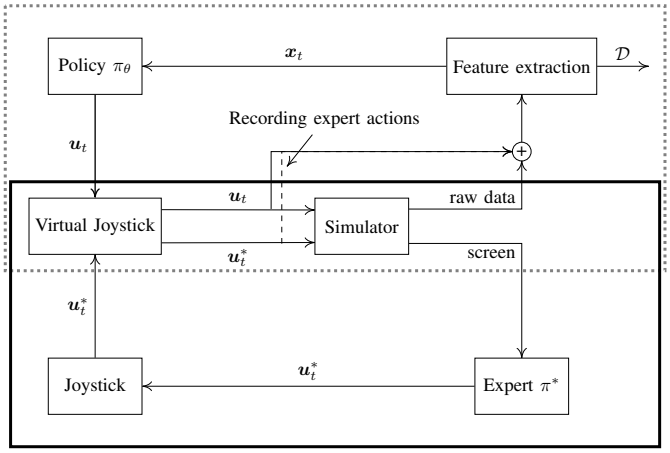


Fig. 3: Illustration of control and data gathering setup. The solid region highlights the human part of the control loop and the dashed region highlights the model part.

A. Tasks

Three tasks are considered:

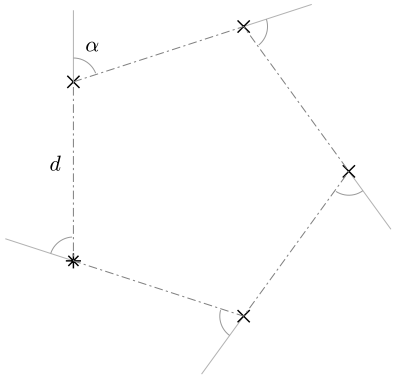
1) Clockwise: Fig. 4a. The first task is to complete as many laps as possible of the pentagon shaped track shown in Fig. 4a. The track runs in a clockwise direction and the star indicates the starting position. Each cross on the track marks a waypoint placed at an altitude of $5000\,ft$ ($1524\,m$). The distance between each waypoints is set to $d = 15\,km$ and the angle is $\alpha = 72°$.

2) Counter-clockwise: This task is the same as task 1, but the track is mirrored about the global y-axis such that the first turn is to the left.

3) Zig-zag: To test a different situation, ten waypoints are placed in a zig-zag pattern as shown in Fig. 4b. Each waypoint is placed according to

$$x_j = x_{j-1} + \lambda_j \sin\alpha_j,$$
$$y_j = y_{j-1} + \lambda_j \cos\alpha_j,$$

where $\alpha_j = 3°, -6°, 9°, \ldots, -30°$ and $\lambda_j = d{\cdot}1.1^{j-1}$ scales the distance such that sharper turns are followed by a longer straight path. This yields a task that increases in difficulty and alternates between left and right turns. In this task, $d = 10\,km$. To complete the track, the agent must perform turns of roughly $3°, 9°, 15°, \ldots, 60°, 70°$.



(a) Task 1

(b) Task 3

Fig. 4: Illustrations of the clockwise and zig-zag flight track. Note that angles of the zig-zag track are exaggerated.

B. State Representation

There are many ways to represent a state to simplify learning with limited amount of data. Since the policy should fly the aircraft on a given track, the information should be invariant to any translation or rotation in planes perpendicular to the gravitational field (assuming close to constant atmospheric conditions). This is intuitive as this policy should consider the same task but translated, for instance, five kilometers north and rotated by 90 degrees as identical and not require new training data. Thus, global quantities are often irrelevant, and only relative changes and local positions are of interest.

Below, four state features are explained with details about their construction to ensure the invariance property previously mentioned. To avoid calculations that treat the motion and curvature of the earth, it is assumed flat and stationary.

1) Tait-Bryan Angles: The policy must be provided with a sense of orientation relative to the surface of the earth. This information is naturally encoded in pitch and roll which are invariant to north/south, east/west directions. Let $\hat{\vartheta}_t = \frac{\vartheta_t}{\pi}$ and $\hat{\phi}_t = \frac{\phi_t}{2\pi}$ denote normalized pitch and roll and let

$$\boldsymbol{\Psi}_t = \begin{bmatrix}\hat{\vartheta}_t, & \hat{\phi}_t\end{bmatrix}^{\mathrm{T}}, \tag{4}$$

be the vector that contains these quantities at time t. Note that yaw is related to the north/south direction and is thus excluded from the state.

2) Tait-Bryan Velocities: The second quantity are rotational velocities, i.e., time-derivatives of the Tait-Bryan angles. Since behavior cloning is used, each visited state is treated as an independent snapshot with a corresponding

action as a label. Consider two seemingly identical situations where the aircraft has the same position and angles. If the aircraft has a fast roll velocity in one of the situations, then at the next time step, the state of the aircraft will have changed significantly and thus require a different action at that time step. If information about the rotational velocity is not included, then the policy cannot discern these two situations, and will compensate for this change in roll angle at a later time step. This would in turn lead to overshooting and possibly yielding an oscillatory behavior, analogous to the derivative-part of a PID-controller [31]. The time-derivatives can be seen as adding a sense of time into the otherwise timeless snapshot. This quantity is denoted, with a slight abuse of notation, as

$$\dot{\boldsymbol{\Psi}}_t = \begin{bmatrix} \dot{\phi}_t, & \dot{\vartheta}_t, & \dot{\psi}_t \end{bmatrix}^{\mathrm{T}}. \tag{5}$$

3) Waypoints: The third quantity is two unit vectors pointing towards the two next waypoints in the local reference frame.

Let

$$\boldsymbol{p}_t = \begin{bmatrix} x_t, & y_t, & z_t \end{bmatrix}^{\mathrm{T}}, \tag{6}$$

and

$$\boldsymbol{w}_j = \begin{bmatrix} x_j, & y_j, & z_j \end{bmatrix}^{\mathrm{T}},$$

denote the aircraft position at time t and the fixed position of the j'th waypoint, both expressed in the inertial reference frame. Define the vector from the aircraft to the waypoint at time t as

$$\boldsymbol{\Delta}_{j,t} = \boldsymbol{w}_j - \boldsymbol{p}_t. \tag{7}$$

As previously explained, it is important to transform information to disregard global quantities. Transforming the waypoint to the local coordinate system can be done using the transformation matrix from equation (3). Denote the transformed waypoint as

$$\boldsymbol{\Delta}'_{j,t} = T(\phi_t, \vartheta_t, \psi_t)\boldsymbol{\Delta}_{j,t}, \tag{8}$$

where the prime-notation indicates a local reference frame. Denote the closest transformed and normalized waypoint to the aircraft in the local forward direction as

$$\hat{\boldsymbol{\Delta}}'_{i,t} = \frac{\boldsymbol{\Delta}'_{i,t}}{\left\|\boldsymbol{\Delta}'_{i,t}\right\|}. \tag{9}$$

Note the new index i, which is defined as

$$i = \arg\min_j \{\left\|\boldsymbol{\Delta}'_{j,t}\right\| : \Delta'_{x,j,t} > 0,\ j \in \mathbb{N}\}, \tag{10}$$

where $\Delta'_{x,j,t}$ denotes the first component of the vector. This index changes when the aircraft has passed the plane parallel to the local xz-plane containing the waypoint. Similarly, one can define the subsequent waypoint with index $i+1$. Because the policy must know how to anticipate the next turn, right or left, information about the two upcoming waypoints are included in the state vector.

4) Distance: Since the waypoint vectors are normalized the distance is lost. To remedy this, the distance to the i'th (current) waypoint is included, divided by the full distance between the waypoints, d. Formally,

$$\hat{d}_{i,t} = \frac{\left\|\boldsymbol{\Delta}'_{i,t}\right\|}{d}.$$

Finally, the full state vector is summarized as

$$\boldsymbol{x}_t = \begin{bmatrix} \boldsymbol{\Psi}_t, & \dot{\boldsymbol{\Psi}}_t, & \hat{\boldsymbol{\Delta}}'_{i,t}, & \hat{\boldsymbol{\Delta}}'_{i+1,t}, & \hat{d}_{i,t} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{12}. \tag{11}$$

C. Action

The behavior cloning method solves a regression problem, yielding a continuous function π_θ that maps states to actions. In this setting, the action consists of three control signals for the aileron, elevator and rudder respectively. Denote the control signal vector at time t as $\boldsymbol{u}_t = \begin{bmatrix} \delta_a, & \delta_e, & \delta_r \end{bmatrix}^{\mathrm{T}} \in [-1, 1]^3$. The neutral position of the joystick is when all values are zero, and the thrust is kept constant, resulting in a cruise speed of roughly Mach 1.3.

D. Preprocessing data

The raw data consists of 6 completed laps (13644 datapoints) on the first track (task 1), flown by the authors. Since this only contains demonstrations of how to fly the track in a clockwise manner, the dataset is augmented with mirrored data to enable flight on the counter-clockwise track. All control signals are shifted as $\boldsymbol{u}_t^{\text{aug}} = \begin{bmatrix} -\delta_a, & \delta_e, & -\delta_r \end{bmatrix}^{\mathrm{T}}$ and for each state the roll, roll velocity, yaw velocity and y-components of the waypoints are multiplied by -1. Assuming that flight dynamic model is perfectly symmetric, this should result in roughly equal performance on both the clockwise and counter-clockwise track.

Much of the training data consists of level flight with small corrections, and less training data with sharp maneuvers. To balance this fact, all datapoints where $\|\boldsymbol{u}_t\| > 0.6$ were kept, and only 50% of the remaining datapoints were kept by sampling uniformly. This value was set after an inspection of joystick distribution.

The full dataset is thus $\mathcal{D} = \{(\boldsymbol{x}_t, \boldsymbol{u}_t) \cup (\boldsymbol{x}_t^{\text{aug}}, \boldsymbol{u}_t^{\text{aug}})\}_{t=1}^{N}$, with $N = 9506$.

Finally, 80% of the dataset was used as a training set and the remaining 20% was used as a validation set. Note that the test set was excluded in this case. The loss value merely indicates that the policy is good at imitating the expert at states induced by the expert, but cannot guarantee well-performing behavior when encountering states not seen in training. Instead, the model is tested in the simulator where the true performance can be measured and analyzed.

E. Training a Policy

The learning policy is represented by a fully connected neural network, illustrated in Fig. 5, trained to map states to actions. The four hidden layers consist of 20 nodes each,

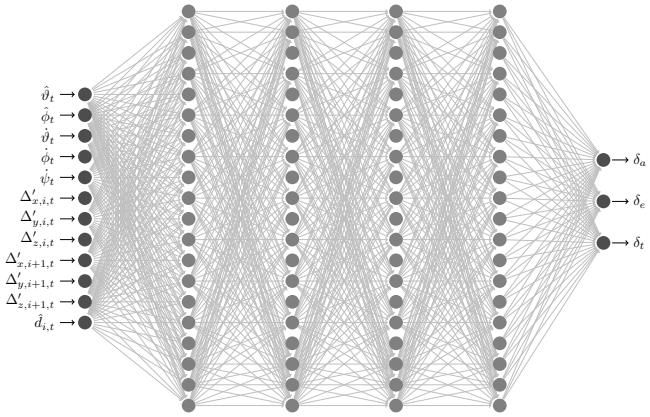


Fig. 5: Illustration of neural network architecture. Note that not all connections are visualized.

using tanh activation functions. The goal is to find the neural network parameters that satisfy,

$$\hat{\theta} = \min_{\boldsymbol{\theta}\in\Theta} \mathbb{E}_{\boldsymbol{x}\sim\rho_{\pi^*}} \left[\mathcal{L}_{\text{MAE}}(\pi^*(\boldsymbol{x}), \pi_\theta(\boldsymbol{x}))\right],$$

where the loss function is the mean absolute error (MAE). The policy was trained using the Adam optimizer, a stochastic gradient decent based method, using the suggested default parameters $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$ [32]. Initial testing showed a significant improvement when using MAE compared to mean squared error (MSE), likely due to this loss function being more robust to outliers [4]. PyTorch [33] was used for implementation and training. The parameters were initialized using Xavier initialization [34] and the full network was trained for 100 epochs with a batch size of 16 and no regularization. The learning rate was a decaying function of epochs, $lr = 0.01 \cdot (0.98)^{\text{epoch}-1}$.

Furthermore, Gaussian noise was added to the input vector at training time, a standard data augmentation technique to reduce overfitting and improve generalization [35]. The noise was constructed such that if the k'th component of the input vector $\boldsymbol{x}_t$ has a standard deviation of σ_k (estimated over the full training dataset), then the new component becomes $\tilde{x}_k = x_k + \frac{\sigma_k}{20}\epsilon$ where ϵ is a random variable from the standard normal distribution. This greatly improved the performance as it artificially augmented the dataset. Since each datapoint was sampled at roughly $10\,Hz$, two neighboring points will be almost identical. Adding noise will thus create a larger number of unique datapoints and force the model to be less sensitive to small perturbations in the state. Naturally, if the noise is too large, the model will not be able to learn.

The network architecture and all hyperparameter values were chosen after a quite extensive trial and error investigation but are not guaranteed to be optimal.

V. Results

The results of the three tasks are presented in this section. Two evaluation metrics were used to evaluate the performance. To capture how well the policy is able to navigate to the waypoints on average, the average minimum passing distance to K waypoints is defined as

$$\overline{d} = \frac{1}{K}\sum_{j=1}^{K} \min_t \|\boldsymbol{\Delta}_{j,t}\|, \tag{12}$$

where $\boldsymbol{\Delta}_{j,t}$ is defined in equation (7). For the expert, this value was $43.95\,m$. The second evaluation metric is an average deviation from expert trajectory τ^*, defined as

$$\overline{d}_{\tau^*} = \frac{1}{T}\sum_{t=1}^{T} \min_s \|\boldsymbol{p}_t - \boldsymbol{p}_s^*\|, \tag{13}$$

where $\boldsymbol{p}_s^*$ denotes the expert position at time s, similar to equation (6). This captures how well the policy is flying, on average, on the same trajectory as the expert. The diligent reader can spot a flaw in this metric. If the learner policy is allowed to randomly visit all states in the near vicinity of the expert trajectory, the metric will be zero. It is assumed, therefore, that the policy is attempting to fly close to the expert trajectory within a finite number of states.

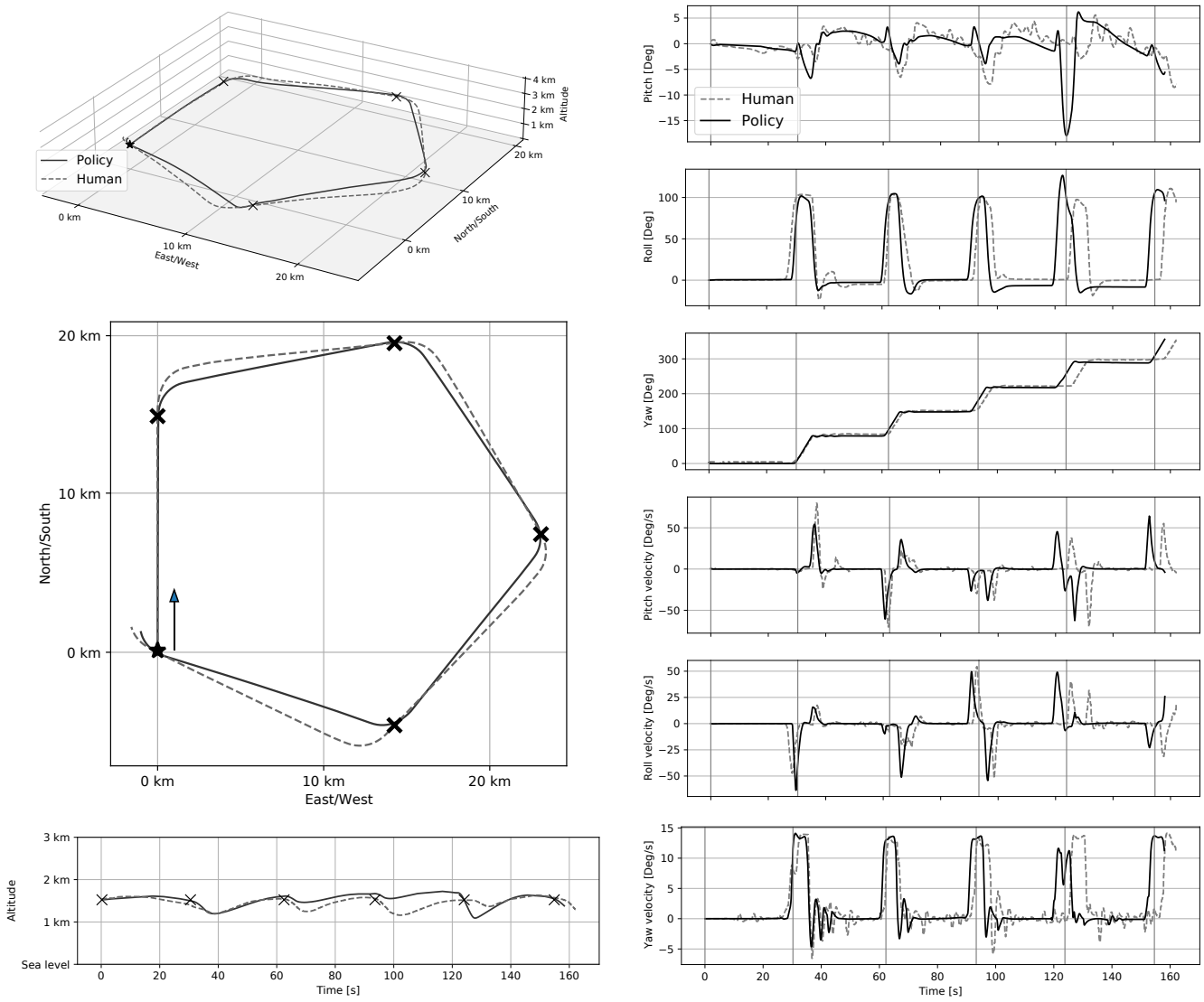
As a performance baseline, the mean deviation was calculated for each of the six expert laps by removing one lap and treating the remaining laps as τ^*. On average, the deviation between each lap was $220.09\,m$, and the max and min was $376.90\,m$, and $149.75\,m$ respectively.

A. Task 1

The trained policy successfully completed the track for nine consecutive laps ($K = 45$ passed waypoints, 20 minutes flight) before the recording was terminated. Fig. 6 illustrates the results by plotting both the policy's and the human pilot's flight trajectories as well as aircraft orientation for one of these laps. Here, it is seen that the results are qualitatively similar with small variations. Most significantly is the dip in pitch after about $125\,s$. The aircraft overshoots the roll angle and in order to compensate, the nose must dip to keep heading towards the waypoints. The policy has also learned a smoothed out version of the pitch. Instead of making small noisy corrections often, it makes larger corrections more seldom. This could probably be improved if the network had more parameters and was trained for more epochs, but it could lead to a more sensitive model that cannot compensate for errors in the same way. It is clear that the policy encounters states that are not a part of the training data, (e.g., $-15°$ pitch and $130°$ roll) but can correct for these states and continue on the path to the next waypoint. Instead of learning the precise actions and certain states, the policy has extracted something more fundamental about how to complete flight towards waypoints and found new solutions to the problem. In addition, the policy starts to turn just before reaching the waypoint to intercept the next waypoint. This was performed on some of the waypoints in the expert data. It is left to the reader to determine if this behavior is desired or not.

In this task, the average minimum passing distance was

$$\overline{d} = 152.49 \pm 8.84\,m,$$



(a) Flight trajectories.

(b) Aircraft orientation and corresponding velocities. Vertical grid lines indicates the passing of a waypoint.

Fig. 6: Flight trajectory (Fig. 6a) and aircraft orientation (Fig. 6b) plots for one lap in task 1. The solid and dashed lines in each plot represent the behavior of the model and the human pilot respectively.

with an estimated 1σ confidence bound using bootstrapping. This corresponds to about $\arcsin\left(\frac{152.49}{d}\right) \approx 0.6^\circ$ deviation from the straight line connecting two waypoints. The average deviation from the expert path was

$$\overline{d}_{\tau^*} = 414.53\,m.$$

B. Task 2

For the counter-clockwise task, the first trajectory is shown in Fig. 7a and the corresponding yaw, pitch roll and their derivatives are shown in 7b. The policy completed 9 laps, same as task 1, before the recording was terminated. In this task, the average minimum passing distance was

$$\overline{d} = 129.21 \pm 4.65\,m.$$

The average deviation from the expert path was

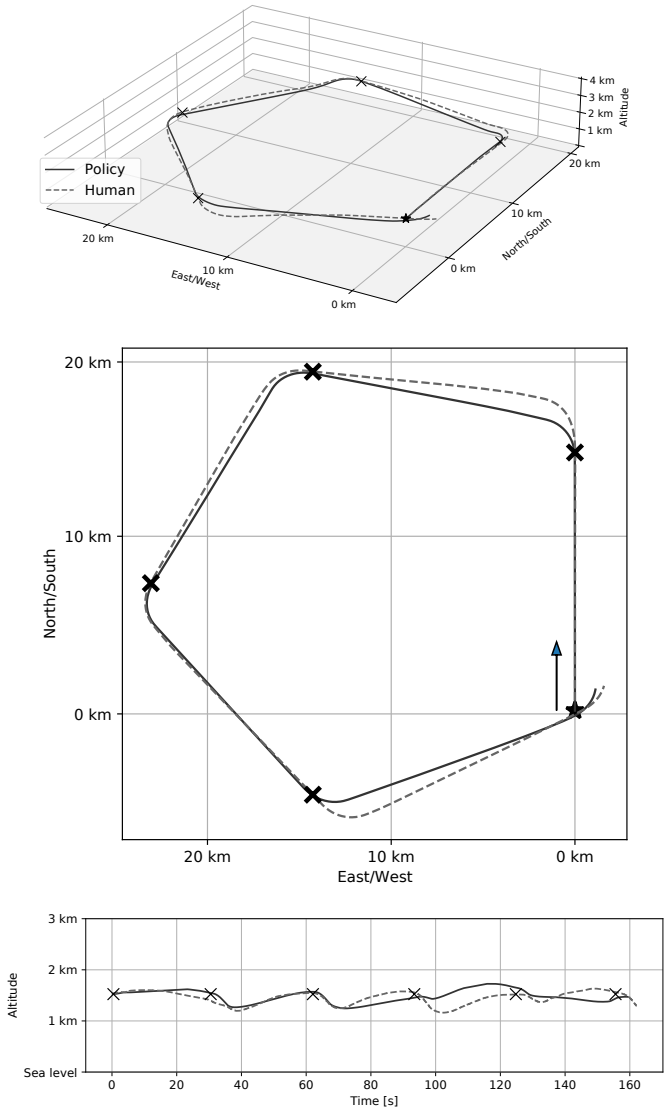
$$\overline{d}_{\tau^*} = 220.17\,m,$$

which is better than in task 1 as can be seen when comparing Fig. 6a with Fig. 7a. This means that augmenting the data set with artificial left turn samples enables the model to complete flight on a track where no expert demonstrations were recorded.

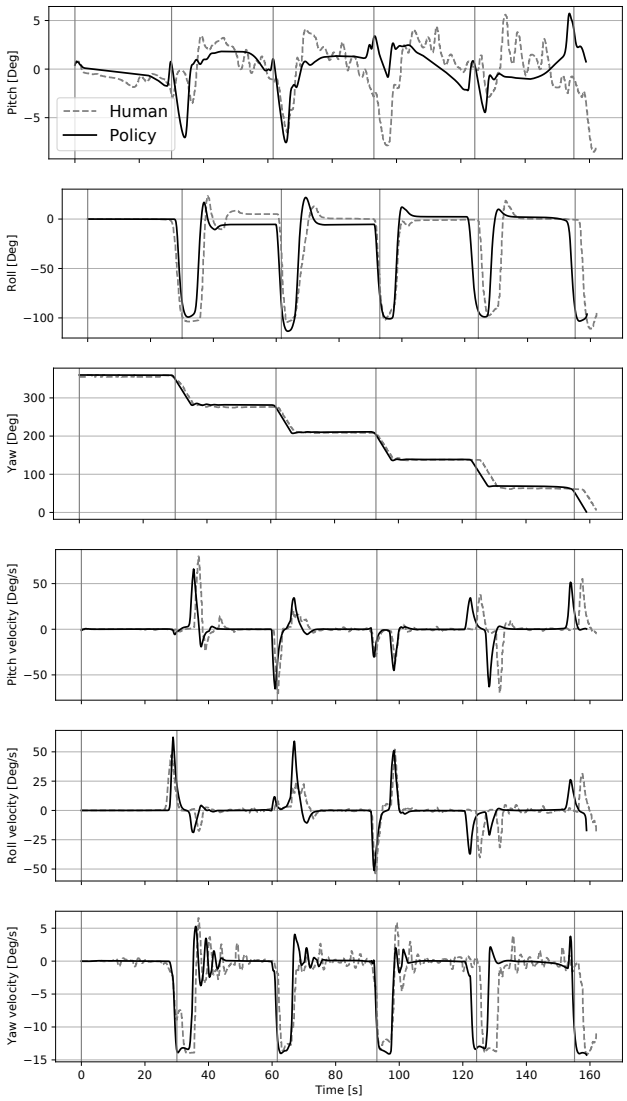
C. Task 3

The resulting trajectories for three attempts at the third task are shown in Fig. 9. For all three attempts, the average minimum passing distance was

$$\overline{d} = 215.85 \pm 15.72\,m,$$



(a) Flight trajectories.



(b) Aircraft orientation and corresponding velocities. Vertical grid lines indicates the passing of a waypoint.

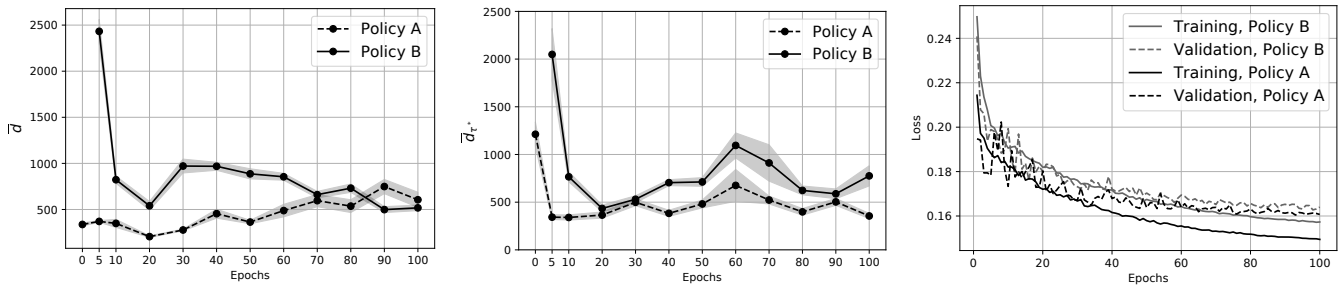
Fig. 7: Flight trajectory (Fig. 7a) and aircraft orientation (Fig. 7b) plots for one lap on task 2. The solid and dashed lines in each plot represent the behavior of the model and the human pilot respectively.

which is slighly higher compared to previous tasks. The policy completes the track in all three attempts, with only slight increase in altitude variation compared to previous tasks. By augmenting the dataset with the mirrored track, the policy was able turn both left and right to complete a flight track where no expert data was recorded. Since no expert data exists, the results for the average deviation from expert trajectory cannot be presented.

VI. Transferring knowledge

Training a neural network means learning a set of parameters that work well for the given task. In this part of the paper, the FDM is changed and the parameters learned on the original FDM are used as a starting point when fine-tuning on target data acquired from flying task 1 with the new FDM. The difference between the two FDMs is difficult to quantify, but for reference, some amount of practice was needed before recording sufficiently good target data.

For clarity, the policy acquired by fine-tuning on the new FDM will be referred to as *policy A*, and a randomly instantiated policy, used for reference, is called *policy B*. A comparison between the two policies is shown in Fig 8. The figure shows the average minimum distance (Fig. 8a), the average deviation from τ^* (Fig. 8b) and the training and validation loss (Fig. 8a), each as a function of training epochs. The average is taken over 3 runs (max 15 waypoints per run) for epochs $5, 10, 20, \ldots, 100$, comparing policy A to policy B.

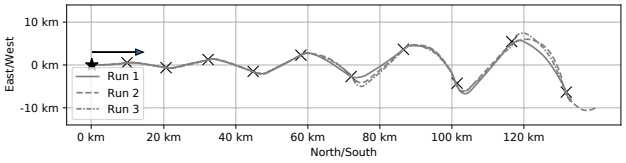


(a) Average minimum distance.

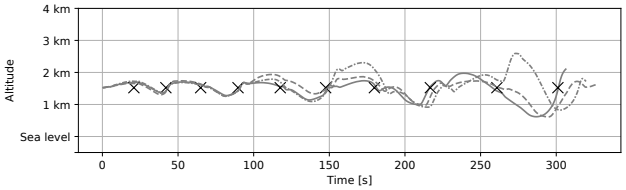
(b) Average deviation from τ^*.

(c) Training and validation loss.

Fig. 8: Average minimum distance to waypoints (Fig. 8a) and average deviation from expert trajectory (Fig. 8b), and MAE-loss (Fig. 8c) as a function of training epochs.



(a) North/South



(b) Altitude/Time

Fig. 9: Resulting trajectories for task 3. Three attempts.

Policy A performs better across all but epoch 90 and 100. At epoch $5, 60$ and 100, policy B only completes 14, 29 and 35 waypoints, whereas the remaining epochs, all 45 waypoints were completed. When policy A has not been fine-tuned at all, it completes 33 waypoints, with a rather low average minimum distance. It is clear that policy A does not improve, and actually gets worse, with more training after epoch 20, even though the loss decreased significantly as seen in Fig. 8c. This illustrates the well-known fact about behavior cloning, that a low training loss does not always imply good performance, since the training and testing distributions are different. This also holds for the validation loss obtained from splitting the training data set, since it stems from the expert distribution ρ_{π^*}, and not from the learner distribution ρ_{π_θ}, which is dependent on the neural network parameters.

VII. Conclusion

In this work, it has been demonstrated that a feed-forward neural network can be trained using behavior cloning. It is shown that satisfactory performance can be achieved on simple flight tasks in a non-linear 6-DOF flight simulator, given that sufficient care is taken during the training process and data preprocessing. Though it is difficult to quantify to what extent the policy is capturing human-like behavior, it is clearly shown that the resulting flight captures much of the behavior found in the (sub-optimal) expert demonstrations, e.g., attempting to roll 90 degrees but overshooting. The policy is also capable of making decisions in states that are not explicitly seen in training, a feature that is not guaranteed by behavior cloning. This is likely due to all state quantities being in the local reference frame, as described in section III-D, which simplified learning. With this transformation, all waypoints seem to be identical from the aircrafts point of view. Instead of training the policy on five different waypoints, six times each, the training data contains 30 identical waypoints. The data is also augmented with an additional 30 waypoints in the other direction that enables turns in both directions. However, due to the limited number of situations in the training data, the model will, not perform well on tasks that are completely different. In order to account for more situations, and produce a more robust policy, one must be active in collecting training data for a diverse set of situations. In addition, it was shown that using a pretrained policy when learning to maneuver the aircraft using a different FDM, leads to better performance with less training. This is desired since it would simplify the development of a large fleet of CGFs that can fly various aircraft.

VIII. Future work

The flight simulator used in this work is primarily intended for other purposes than training and executing machine learning models. It has been a challenge to develop a setup than can log data and deploy neural networks able to control the virtual joystick. A major downside is that the simulations only operate in real-time, so each test run is time-consuming and scaling the number of experiments to get statistically accurate measurements is infeasible. Using a simulator with a graphics interface that can work in real- and simulation-time would allow for significant improvement in both training and execution of the models.

Another extension to this work would be to reproduce the same experiments using a physical aircraft. This would likely highlight problems not present in the simulator, such as noise, latency and wind. It would be of interest to explore transfer learning scenarios from simulation to reality (sim2real) and determine if simulated data can improve the performance of a policy executed into the real world.

Combining more tasks with a more robust and thorough analysis method to determine to what extent the models are actually imitating real human pilot behavior in terms of, for instance, reaction time, likelihood of making incorrect decisions etc, are interesting future research directions. Last but not least, assuming that the models become human-like in their behavior, it is of interest to transfer these models to real training facilities and to evaluate the benefit it has for pilots.

ACKNOWLEDGMENT

This work was supported by the FOI research projects "AI for decision support and cognitive systems" and "Development of the Swedish Air Force Combat Simulation Centre", which are funded by the R&D programme of the Swedish Armed Forces.

REFERENCES

[1] A. Toubman, G. Poppinga, J. J. Roessingh, M. Hou, L. Luotsinen, R. A. Løvlid, C. Meyer, R. Rijken, and M. Turcanık, "Modeling CGF behavior with machine learning techniques: Requirements and future directions," in *Proceedings of the 2015 Interservice/Industry Training, Simulation, and Education Conference*, 2015, pp. 2637–2647.

[2] A. P. Pope, J. S. Ide, D. Micovic, H. Diaz, D. Rosenbluth, L. Ritholtz, J. C. Twedt, T. T. Walker, K. Alcedo, and D. Javorsek, "Hierarchical reinforcement learning for air-to-air combat," *arXiv e-prints*, May 2021.

[3] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," Carnegie-Mellon University, Tech. Rep., 1989.

[4] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," 2018.

[5] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010.

[6] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011.

[7] D. Shukla, S. Keshmiri, and N. Beckage, "Imitation learning for neural network autopilot in fixed-wing unmanned aerial systems," in *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2020.

[8] A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat, and J. Van Den Herik, "Dynamic scripting with team coordination in air combat simulation," in *International conference on industrial, engineering and other applications of applied intelligent systems*. Springer, 2014, pp. 440–449.

[9] A. Toubman, J. Roessingh, P. Spronck, A. Plaat, and H. van den Herik, "Improving air-to-air combat behavior through transparent machine learning," 2014.

[10] J. Källström and F. Heintz, "Agent coordination in air combat simulation using multi-agent deep reinforcement learning," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2020, pp. 2157–2164.

[11] X. Ma, L. Xia, and Q. Zhao, "Air-combat strategy using deep Q-learning," in *2018 Chinese Automation Congress (CAC)*, 2018, pp. 3952–3957.

[12] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[13] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end autonomous driving," *arXiv e-prints*, p. arXiv:1605.06450, May 2016.

[14] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. A. Theodorou, and B. Boots, "Agile autonomous driving using end-to-end deep imitation learning," in *Robotics: Science and Systems*, 2018.

[15] W. Farag, "Cloning safe driving behavior for self-driving cars using convolutional neural networks," *Recent Patents on Computer Science*, vol. 12, no. 2, 2019.

[16] T. Pearce and J. Zhu, "Counter-Strike deathmatch with large-scale behavioural cloning," *arXiv e-prints*, Apr. 2021.

[17] C. Sammut, S. Hurst, D. Kedzier, and D. Michie, "Learning to fly," in *Machine Learning Proceedings 1992*. Elsevier, 1992.

[18] H. Baomar and P. J. Bentley, "An intelligent autopilot system that learns piloting skills from human pilots by imitation," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2016, pp. 1023–1031.

[19] P. J. Bentley and H. Baomar, "An intelligent autopilot system that learns flight emergency procedures by imitating human pilots," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2016, pp. 1–9.

[20] K. Sezginer and C. Kasnakoğlu, "Autonomous navigation of an aircraft using a NARX recurrent neural network," in *2019 11th International Conference on Electrical and Electronics Engineering (ELECO)*, 2019, pp. 895–899.

[21] H. Baomar and P. J. Bentley, "Autonomous navigation and landing of large jets using artificial neural networks and learning by imitation," in *2017 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 2017, pp. 1–10.

[22] J. Pinguet, P. Feyel, and G. Sandou, "A neural autopilot training platform based on a Matlab and X-Plane co-simulation," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2021.

[23] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[24] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" *Advances in Neural Information Processing Systems*, vol. 27, pp. 3320–3328, 2014.

[25] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *The journal of machine learning research*, vol. 17, no. 1, 2016.

[26] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018.

[27] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *International conference on machine learning*. PMLR, 2016, pp. 1842–1850.

[28] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing, and R. Feris, "Spottune: transfer learning through adaptive fine-tuning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4805–4814.

[29] B. Etkin and L. D. Reid, *Dynamics of Flight*. Wiley New York, 1959, vol. 2.

[30] P. Flores, "Euler angles, Bryant angles and Euler parameters," in *Concepts and formulations for spatial multibody dynamics*. Springer, 2015.

[31] A. Visioli, *Practical PID control*. Springer Science & Business Media, 2006.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019.

[34] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010.

[35] G. An, "The effects of adding noise during backpropagation training on a generalization performance," *Neural computation*, vol. 8, no. 3, pp. 643–674, 1996.